

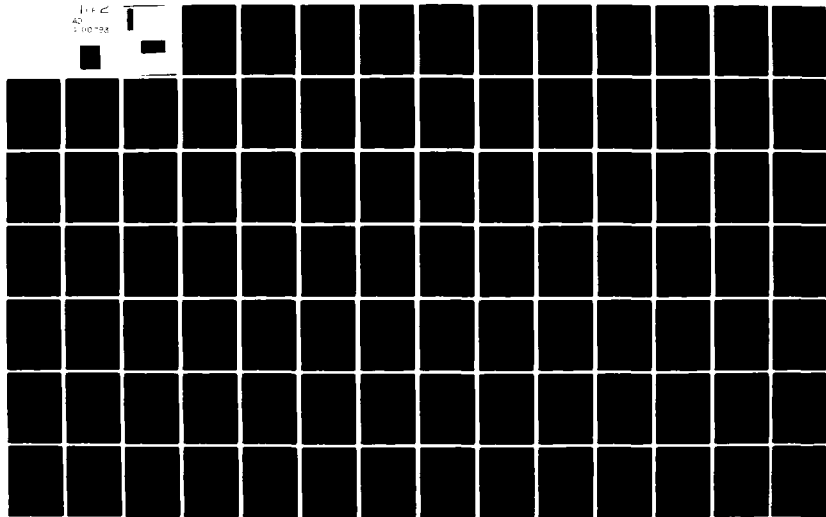
AD-A100 793

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCH00--ETC F/6 9/2
DESIGN OF A RESOURCE-SHARING NETWORK LINK.(U)
DEC 80 T M MCLEOD
AFIT/6E/EE/80D-30

UNCLASSIFIED

NL

1-1-2
40
1-10-79



AFIT/GE/EE/80D-30

①

① Dec 80

① 12 1-1

①
DESIGN OF A
RESOURCE-SHARING NETWORK LINK.

①
THESIS

AFIT/GE/EE/80D-30 ① Thomas M. McLeod
Capt USAF

Approved for public release; distribution unlimited

012225

DESIGN OF A RESOURCE-SHARING NETWORK LINK

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology

Air Training Command
in partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Thomas M. McLeod, B.S.E.E.

Capt

USAF

Graduate Electrical Engineering

December 1980

ng
A

Approved for public release; distribution unlimited

Preface

This report presents the analysis and design of a resource-sharing computer network link. The work described in this report was performed in support of the Automated Management System (AMS) which is being development by the Aeronautical Systems Division (ASD). It is hoped that the design will result in added capabilities for the existing network in the near future and that it will also aid planning efforts for full scale development of the AMS Network.

I wish to acknowledge several individuals whose support made this thesis possible. Professionally, I owe the greatest debt to Dr. Gary Lamont, my thesis advisor, for his guidance and understanding. Dr. Lamont's skillful guidance resulted in the thesis effort being enjoyable as well as educational. My thesis sponsor, Capt Chris Allen of the ASD Computer Center, was a constant source of support for which I am very grateful. I wish to thank my thesis committee, Lt Col Rutledge and Capt Seward, for their cooperation and support.

Most of all, I wish to extend heartfelt praise to my wife, Carol, and children, Edward, Patrick, and Shannon, for the dignified way that they have endured a difficult period in our lives.

Contents

	Page
Preface	11
List of Figures	vi
Abstract	viii
I. Introduction	1
AMS Definition	2
Network Fundamentals	3
AMS Computer Network	7
Present AMS Configuration	11
Scope of investigation	11
Approach	12
Order of Presentation	13
II. System Requirements Analysis and System Design . .	14
System Requirements	14
System Requirements Analysis	15
Requirements Analysis of R-S Functions . . .	16
Transfer Sequential File	18
Transfer Program File	20
Transfer Message	22
Execute Remote Program	24
Execute Remote Program Development	26
System Requirements Analysis Summary	28
Existing AMS Modules	28
Model 990 Computer	28
Peripherals	29
Communications Interface Card	30
3780 Emulator	31
DX-10 Operating System	32
Operating System Interfaces	33
Operator/Operating System	33
Applications Program/Operating System	36
Operator/Applications Program	36
Applications Program/Applications Program . .	37

	Page
Applications Program/SCI	37
System Design	40
Network Commands	45
Summary	46
III. Software Requirements Analysis	48
Network Initialize (NI)	54
Network Log On (NLON)	56
Network Transfer File (NTF)	58
Network Message Commands	61
Network Message Compose (NMC)	61
Network Message Send (NMS)	63
Network Message Abort (NMA)	63
Network Transfer Program File (NTPF)	63
Network Execute Task (NXT)	70
Network Compile (NCOMP)	72
Network Link Edit (NLINK)	74
Network Execute Batch (NXB)	76
Network Log Off (NLOFF)	78
Network Quit (NQ)	80
Summary	82
IV. Network Software Design	83
Structured Design	83
Overview	85
O/S Interface Procedures	89
Putmsg	90
Getmsg	90
Delay	90
Read and Write	90
Call SCI	91
Findsyn	91
Storesyn	91
Taskid	92
FORENET	92
Get Comd	93
Get Parm	93
Initialize Link	94
Execute BN2	94
Initialize Emulator	94
Log On	95
Send Sequential File	95
Receive Sequential File	95
Execute Task	95

Execute Batch	96
Compile	96
Link Edit	96
Log Off	96
Terminate Emulator	96
BACKNET	97
Get Comd	97
Get ParmS	97
Initialize Link	97
Log On	97
Send Program File	98
Receive Program File	98
Send Sequential File	99
Receive Sequential File	99
Execute Task	99
Execute Batch	99
Compile	100
Link Edit	100
Log Off	100
Summary	100
V. Results and Recommendation	101
Design results	101
Recommendations	103
Bibliography	105
Appendix	107
Vita	119

List of Figures

Figure		Page
1	Typical Network Configuraion	4
2	Protocol Hierarchy	8
3	AMS Computer Network	10
4	Resource-Sharing Link	17
5	Transfer Sequential File	19
6	Transfer Program File	21
7	Compose and Abort Message	22
8	Send Message	23
9	Execute Remote Program	24
10	Compile and Link Edit	27
11	Operating System Relationships	34
12	System Design	42
13	FORENET and BACKNET R-S Processes	43
14	Typical Command	50
15	User Process/R-S Process Interface	53
16	Network Initialize (NI)	55
17	Network Log On (NLON)	57
18	Network Transfer File (NTF)	59
19	Network Message Commands	62
20	Network Transfer Program File (NTPF)	65
21	Send Program File	67
22	Receive Program File	68
23	Network Execute Task (NXT)	71

Figure		Page
24	Network Compile (NCOMP)	73
25	Network Link Edit (NLINK)	75
26	Network Execute Batch (NXB)	77
27	Network Log Off (NLOFF)	79
28	Network Quit (NQ)	81
29	Structure Charts	84
30	FORENET Structure Chart	87
31	BACKNET Structure Chart	88

Abstract

A resource-sharing (R-S) computer network link design is presented. The design complies with requirements for the Automated Management System (AMS) being developed by the Air Force's Aeronautical Systems Division (ASD). The design includes Texas Instruments Model 990 computers that currently exist in the AMS Network.

R-S software was designed that binds existing modules into, effectively, one R-S process at each end of the link. The two R-S processes cooperate in accordance with R-S protocols to accomplish the specified R-S functions. Existing capabilities of the DX-10 operating system were used extensively. The processes communicate via telephone circuits using TI 3780 Emulator software and standard data communications hardware. The R-S functions implemented include file transfer, program file transfer, and remote program execution, as well as others.

A unique interface to the DX-10 operating system was designed to allow the R-S software to have adequate access to the operating system. The operating system batch command was emulated, allowing the R-S program to construct a "batch file" containing any operating system commands that would normally be entered from a interactive terminal.

I. Introduction

As the benefits of computers in the business and office environment have become more widely known, the demand for computer-based management information services has increased dramatically. Management information applications usually involve large amounts of data and specialized computer programs are required to analyze the data. However, many of these applications are beyond the scope of the mini-computer systems that can be economically utilized in the office environment. Fortunately, the rapidly advancing computer networking technology provides a solution. Networking allows the resources of the larger, more versatile, computer systems to be shared with the smaller office computers.

The Air Force's Aeronautical Systems Division (ASD) is planning the development of the Automated Management System (AMS) which will provide a large variety of management information services to the managers of Air Force weapons system development programs. The services will include the maintenance of large data bases, analysis of data bases, and the creation of reports and graphical presentations. AMS will consist of 30 small computers systems and at least 5 large computer systems. The computers will be interconnected by a data communications subnetwork to create

the AMS Computer Network. The networking of the computers will allow each site to share the resources of the other computers in the network. The objective of this investigation was to take the first step toward realizing the AMS resource-sharing (R-S) computer network by designing a single R-S link. The link was constrained to consist of two Texas Instruments Inc. Model 990 computer systems.

AMS Definition

The ASD Automated Management System (AMS) is an evolving, computer-based management information system that aids managers of Air Force weapon system development programs by storing and analyzing management data. The objective of AMS (Ref 1) is to provide management information services to 30 user locations known as sites. AMS will allow managers direct access to the provided services. The ease of operation will be achieved by defining a minimum number of interactive commands to effect the desired services. Ideally, only one command would be needed for a frequently requested service. Additionally, any parameters required with a command will be minimized and the operator will be prompted for the parameters in unambiguous English.

Since each site is a computer system, many of the simpler management information services can be provided autonomously by the site. For instance, the accounting of

office travel funds is a very localized service that requires limited storage and relatively simple processing, and is, therefore, within the capability of a site. However, the analysis of cost data for an aircraft development program might require the resources of a large computer system. The resources of large, remote computer systems will be available through the AMS Computer Network. Before describing the AMS Computer Network, a brief tutorial of computer networks is presented to familiarize the reader with the terminology and concepts used in this investigation.

Network Fundamentals

Figure 1 illustrates a typical network topology. In the figure, squares represent hosts, circles represent nodes, and lines represent physical links. Hosts are user computers that are connected to other user computers via the network. Nodes are components of the data communications subnetwork and provide routing and multiplexing functions. The subnetwork is the portion of the network that logically ties the hosts together. The interface between a host and the subnet is usually internal to the host since the host often performs some of the subnetwork data communications functions.

For the hosts and nodes to communicate with each other, they must complement each other physically and logically.

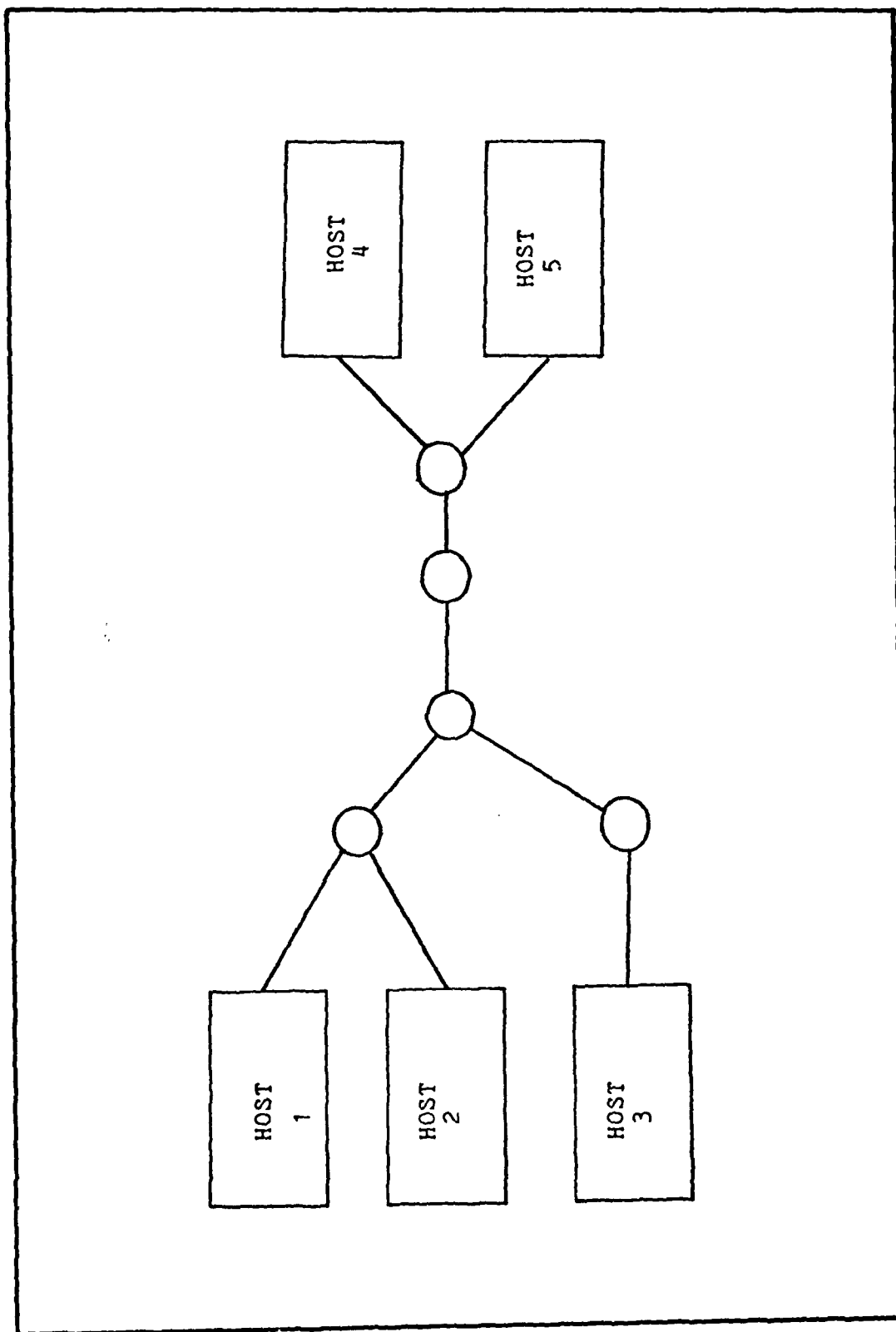


Figure 1. Typical Network Configuration

The participants in the network must adhere to communications conventions, or protocols. The protocols are divided into layers, or levels, of functions for ease of understanding, implementation, and standardization. The levels defined by one particular partitioning scheme (Ref 21:66-67) are described in the following subparagraphs.

Physical. The physical level protocols establish a physical and electrical communications path. The physical level is considered to be the lowest level since it provides the fundamental communications path used by the higher level protocols.

Link. Above the Physical level protocol is the link level protocol which provides error control and some flow control. This layer's primary function is error protection which it provides by constructing the data into frames and adding error detection and correction code bits. Since the data is divided into frames, a limited flow control can be implemented by labeling the frames and keeping track of which frames in the sequence have been successfully transported. When an error is detected, the flow can be stopped while corrective action is taken between network elements.

Network. Next, the network level protocols route and multiplex the data frames, making sure that each frame gets

to the next proper node on its way to its proper destination. Network protocols concatenate links to establish a logical path between two hosts.

Transport. Above the network level, the transport level controls the flow of messages between two hosts. The transport level assumes a virtual link exists between the hosts. In other words, the transport level is "unaware" of the routing functions being accomplished by the network level. If there are several process pairs communicating between the two hosts, the transport layer provides routing and multiplexing for those process pairs.

Session. The session layer establishes and maintains the identities of the communicating processes. In other words, the session layer establishes communications between a unique pair of processes.

Presentation. The presentation layer insures that the data structures being used between two processes are compatible. For example, if it is necessary, the presentation layer will convert a file to a common subnetwork format for transmission. At the destination the presentation layer converts the common format to the local format.

Application. The highest level is the Applications, or Resource-Sharing, level. The applications level is the only layer that serves the user directly. The most elementary R-S functions are file access and file transfer. Other

services such as batch processing and remote terminal support are included as well. All of the lower level protocols exist to support the R-S level.

The protocols defined above are related to each other as shown in figure 2. The user data enters at the R-S level. The data traverses the network as indicated by the line that threads through the levels, eventually connecting the R-S level of two hosts.

The protocol levels are implemented as hardware and software processes. A process corresponding to a protocol level will interact only with a complementary process that implements the same protocol level. The horizontal lines represent the process interactions. The processes at each level are not "aware" of lower level processes. The fact that lower level protocols are transparent is a major criterion and advantage of layering. As a result, lower level protocols can be replaced or changed without affecting the operation of a higher level protocol as long as the interface with the adjacent lower level process does not change.

AMS Computer Network

The AMS R-S Computer Network will connect at least 5 large computer systems and all of the sites, including a special site know as the Software Development and Maintenance Facility (SDMF). In the context of the Network

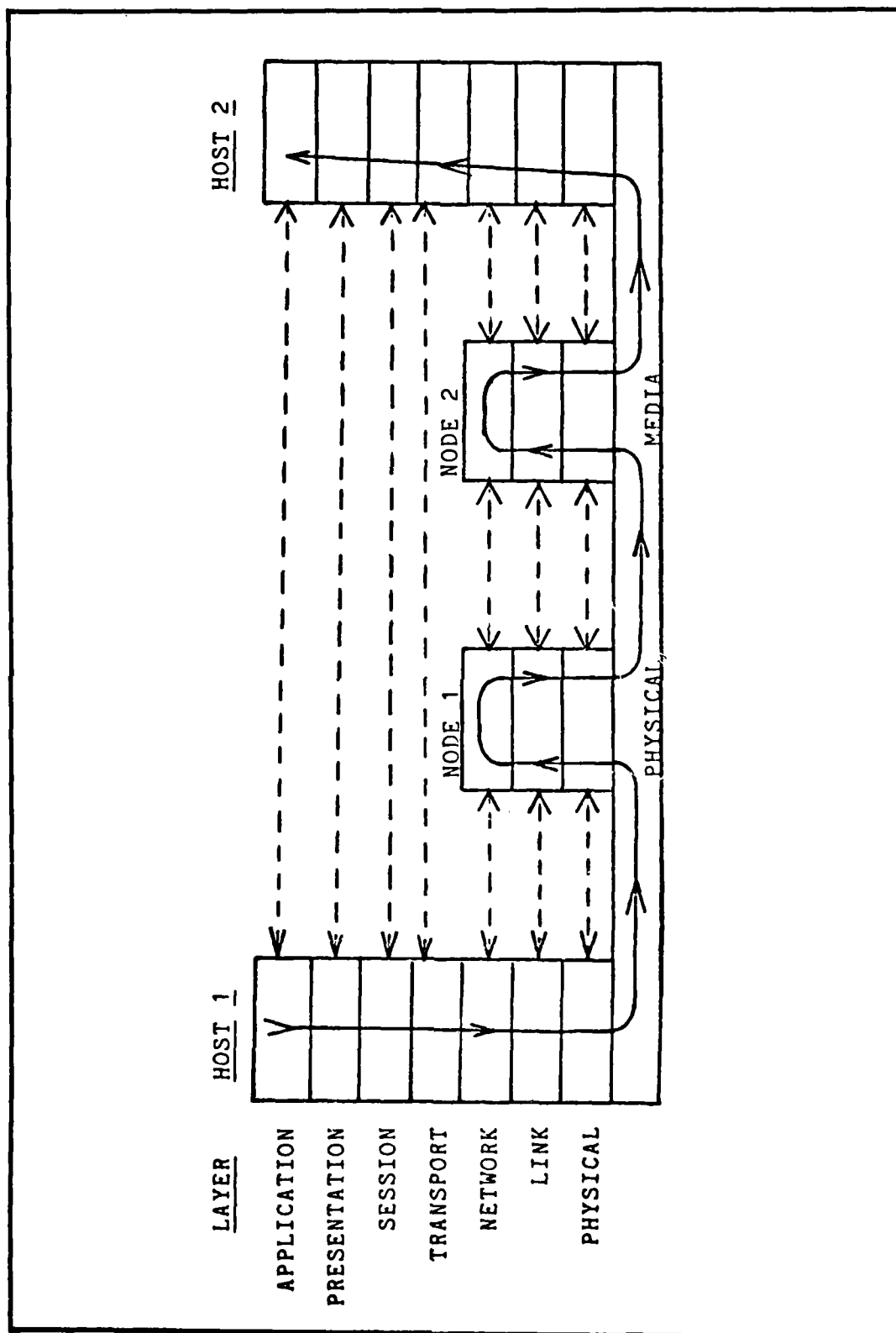


Figure 2. Protocol Hierarchy

Fundamentals section, the sites would be known hosts, since they are, in fact, user computers served by the network. In the context of AMS, however, they are classified as sites to differentiate them from the larger computer systems that are known as hosts. The planned configuration of the network is shown in Figure 3. All hosts and sites are connected directly to a single node, forming a "star" network topology.

The AMS node routes and multiplexes data communications as was the case in the general network configuration. However, the AMS node also translates the AMS commands from the sites into the job control language (JCL) for each host. A JCL is a set of commands used to supply information to a computer's operating system and to request resources under the control of a computer's operating system. A uniform, simple command set that eases the AMS operators workload is only possible if the JCLs of the hosts can be created automatically by AMS.

The SDMF is a special site that is configured for efficient program development activities. It will be the same basic computer type as the other sites, but it will have increased primary and secondary memory capacities, and program development tools such as editors, assemblers, compilers, and debuggers (Ref 18).

Each site can be uniquely configured, but the typical

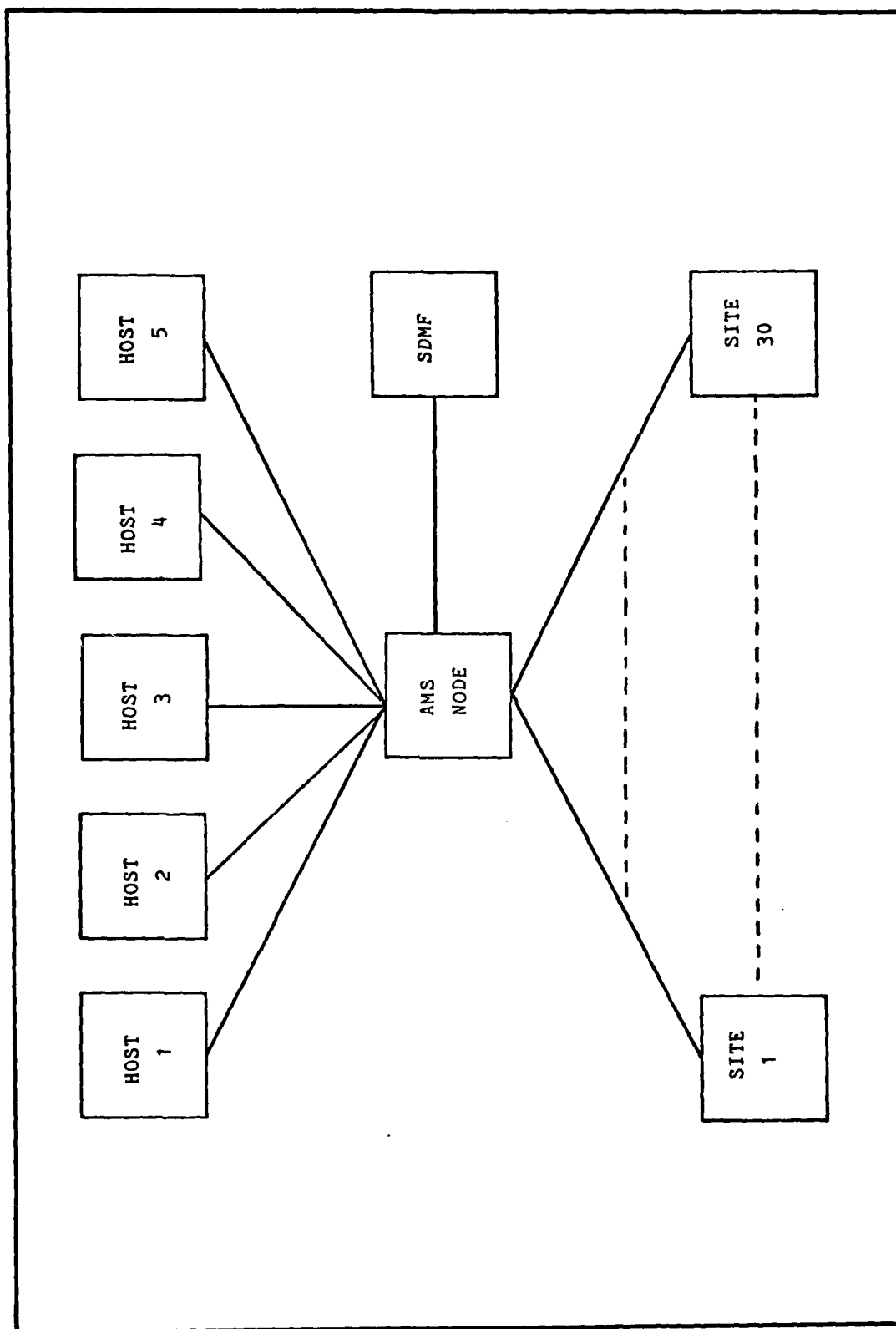


Figure 3. AMS Computer Network

site will consist of a small disk memory (5 Megabytes) and two video display terminals (VDT).

Present AMS Configuration

Since AMS is an "evolving" management information system (MIS), some of the AMS goals have been implemented in a prototype system. The present AMS consists of 3 hosts, 3 sites, and the SDMF. Some communications capabilities exist and several MIS programs are operational.

Some data communications capabilities exist in AMS, but the computers are not actively networked. Each site communicates with a host individually much the same way one would interface with a large computer from a time sharing terminal. A computer program has been developed that allows a site to communicate with a host using asynchronous teletype protocols. Another program is under development to translate an AMS language into a partial set of the JCL commands of the 3 hosts (Ref 5).

Several MIS programs such as one that acquires and processes travel fund data are operational. Many other MIS programs are currently under development.

Scope of Investigation

The objective of this investigation was to design a resource-sharing link between two Texas Instrument's Model 990 computers. The R-S link design objectives (Ref 3)

specified the following capabilities:

Sequential File Transfer

Relative Record File Transfer

Intersite Message Transfer

Remote Program Execution

Remote Program Development Capability

These requirements are a subset of the overall AMS Network requirements. The primary difference between the two sets of requirements is that the routing and multiplexing requirements have been omitted. Additionally, the existing components of AMS were to be used to the maximum extent possible and any software required would be designed for implementation in the Pascal language. Using existing components will reduce the amount of time to realize a resource-sharing capability for AMS. Pascal was chosen by the user because it is a structured language that results in programs that are easier to read and, therefore, easier to maintain. It is also hoped that the Pascal language will reduce the development effort.

Approach

The investigation began with an analysis of the AMS R-S Network Link Requirements. The purpose of the analysis was to determine the functions necessary to realize an R-S link capable of satisfying the R-S link system requirements.

Having defined the basic R-S functions that would be

required to realize the R-S link, the existing AMS hardware and software modules were evaluated for their applicability. Next, a system design was accomplished using applicable existing modules and, as necessary, newly defined modules. All of the new modules were software, resulting in the remainder of the investigation concentrating on the development of software.

The software development began with a structured analysis of the software requirements to identify the processes that would be implemented in software. Data flow diagrams were used to illustrate the processes. The defined software processes were next allocated to software modules using structured design techniques. The resulting modules were illustrated with structured design diagrams.

Order of Presentation

Chapter 2 describes the analysis of the system requirements and also describes the system design. Chapter 3 presents the software requirements analysis, including the data flow diagrams. Chapter 4 describes the design of the necessary software programs and includes the structured design charts that illustrate the design. Chapter 5 contains results and recommendations. The appendix describes a special operating system interface developed by this investigation and contains program listings used to demonstrate and test the interface.

II. System Requirements Analysis and System Design

This chapter describes the analysis of the AMS Resource-Sharing (R-S) Computer Network Link system requirements and the design of the R-S link. First the system requirements were analyzed to determine the processes which must be implemented in either hardware or software. Next, existing hardware and software modules were evaluated for their applicability to the system design. Finally, a system design was accomplished that consists of existing hardware and software modules, and newly defined software modules.

System Requirements

The basic requirement for the AMS R-S computer network link was to create a R-S network link between two Model 990 computers that permits an operator or applications process at either end of the link to utilize the resources of both computers. In addition to this basic requirement, it was required that the network link design be consistent with the overall AMS goals. The primary AMS goal that impacted the link design was the goal that AMS be simple to operate.

The design was constrained to use existing hardware and software modules to maximum extent possible to reduce the time required to implement the R-S link(Ref 3). Additionally, the following R-S functions were required as a

minimum:

Transfer Sequential Files

Transfer Program Files

Transfer Intersite Messages

Execute remote programs

Execute remote program development

To achieve the goal of making AMS easy to operate, it was required that the AMS commands be limited to one per basic R-S function and that parameters associated with each command be limited to those which can not be deduced by the system from other information sources. The interface to an applications program had to be correspondingly simple to minimize applications programming effort by the network user.

System Requirements Analysis System requirements analysis is a methodology for determining what functions, or processes, a system must perform to satisfy the system requirements. In the context of requirements analysis, the word process has a specific meaning. A process is a logical entity, described by a verb-object phrase, that transforms some other entity, such as data. A process represents the action that must be performed but does not specify how it will be implemented. The general requirement for an R-S link was analyzed first and then the specific R-S functions were analyzed.

Figure 4 illustrates the most basic processes associated with a R-S link. In general, there would be a user process and two R-S processes. One of the R-S processes would be at the local, or initiating end, and the other R-S process would be at the remote end. The using process generates R-S commands to request particular R-S services and the R-S processes cooperate to execute the functions indicated by the commands. The R-S processes must cooperate in accordance with a strict set of rules, or protocols to achieve all data communications functions from the link level to the R-S level. Protocols enable the two independent, concurrent R-S processes to synchronize their activities. Each process proceeds through a set of logical states which are determined by communications between the processes.

Requirements Analysis of R-S Functions. The major processes needed for each R-S function are described in this subsection. The processes are illustrated with Data Flow Diagrams (Ref 11). A Data Flow Diagram (DFD) uses circles and connecting lines to represent processes and interprocess data flow, respectively. Protocols between the processes are not apparent in DFD's. Protocols determine the flow of control between the processes and DFD's only show flow of data between the processes. The specific protocols

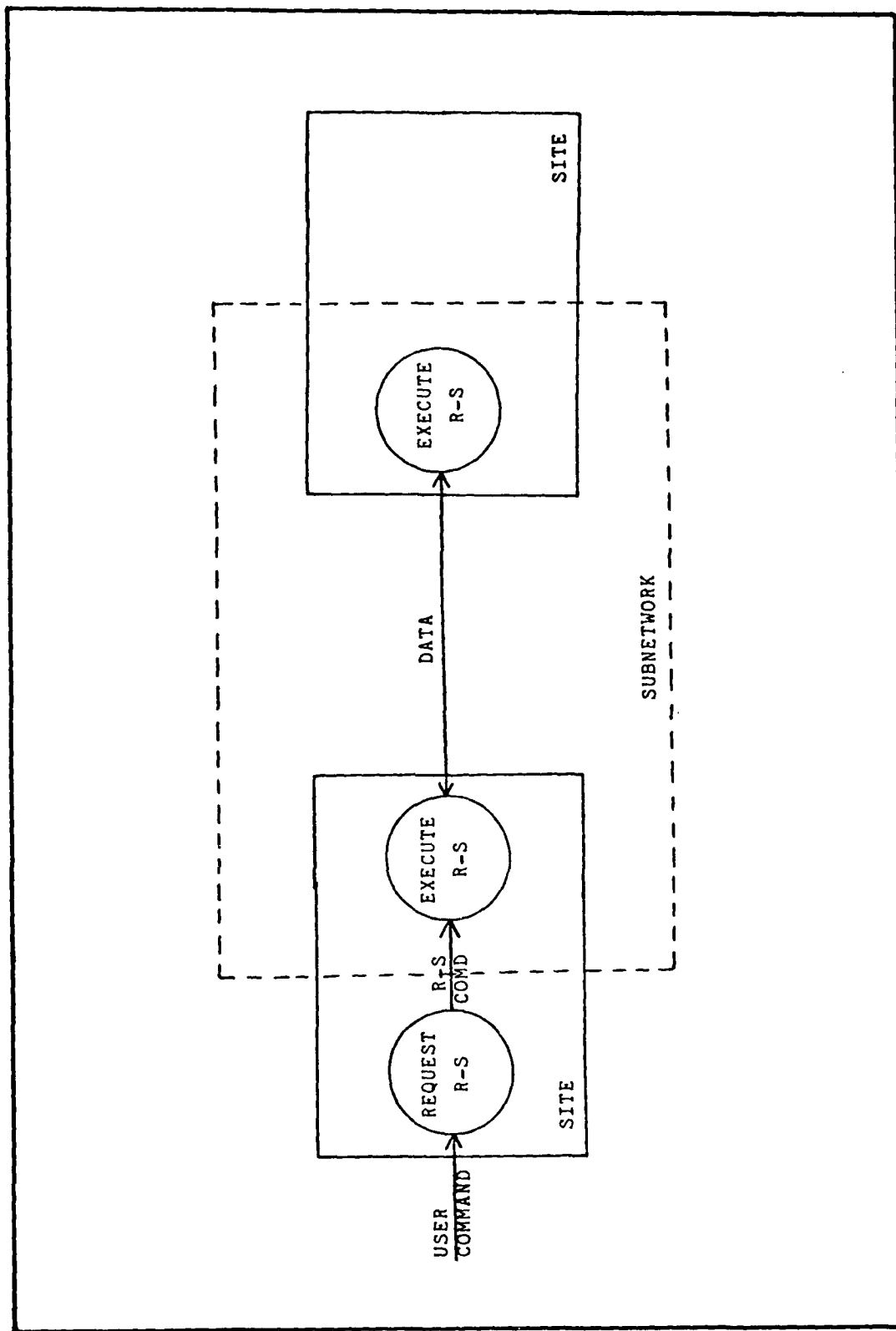


Figure 4. Resource-Sharing Link

necessary to implement the R-S functions will be described in the next chapter.

Transfer Sequential Files. The transferring of sequential files is the most elementary and most common R-S function. Sequential files are typically data files, and, in a R-S network, one would often want to transfer a data file to a remote computer for processing by a program resident in that computer. A sequential file could also be a computer language source file or a text file. The DFD in figure 5 shows the basic processes required to transfer a sequential file. The status message can convey many types of progress information such as acknowledgement and error. The termination message would vary also. The termination messages inform the user of a successful or unsuccessful execution and provides additional information to aid recovery if the execution is unsuccessful.

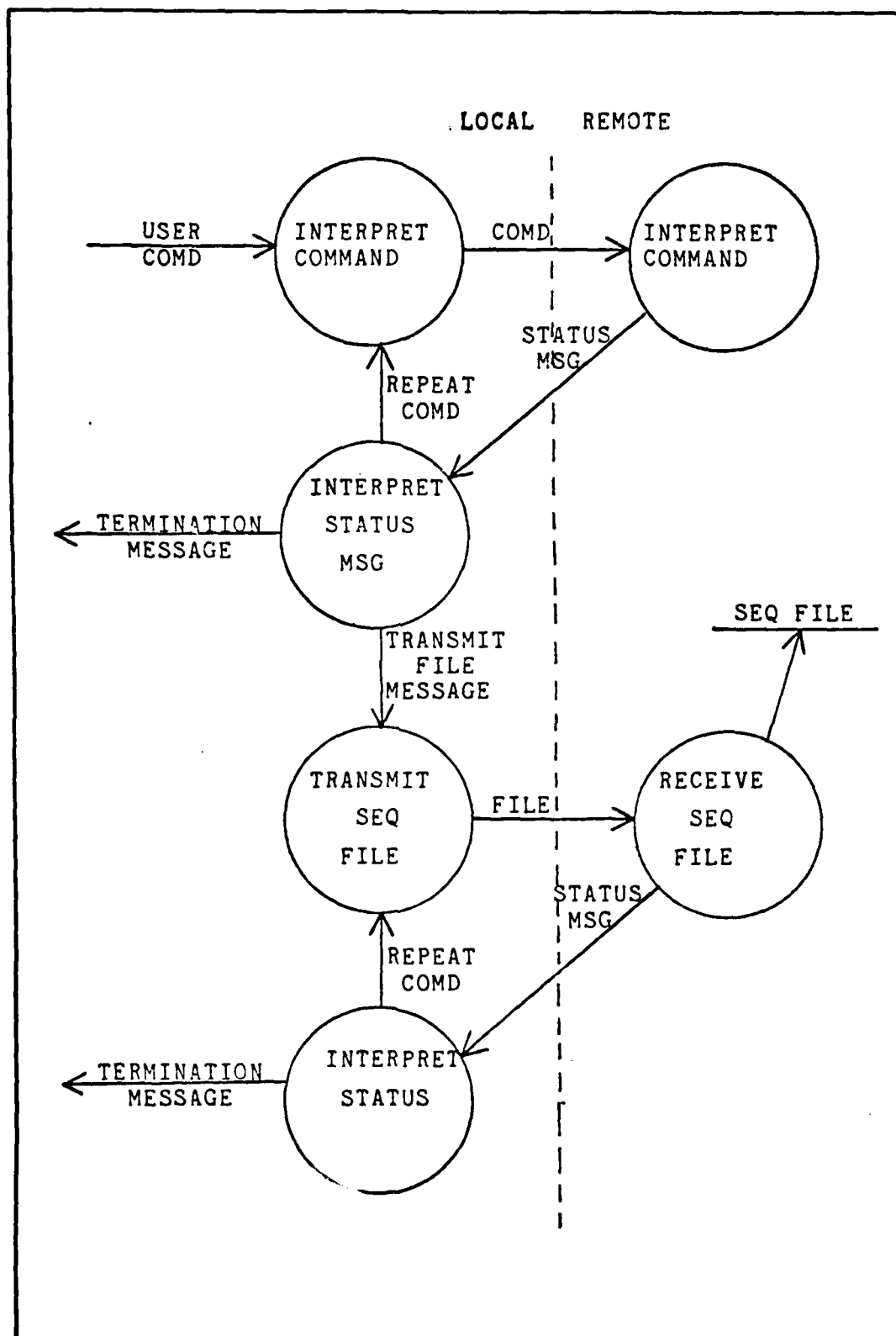


Figure 5. Transfer Sequential File

Transfer Program File. The transferring of a program file is handled differently than a sequential file. A program file is relative record, or random, file. A program file contains several executeable object code programs. The DFD in figure 6 shows the processes required to transfer a program file. The program file is converted to a sequential file prior to transmission and back into a program file when it arrives at its destination. The formatting prior to transmission simplifies and standadizes the file format that must be processed by the lower level protocols.

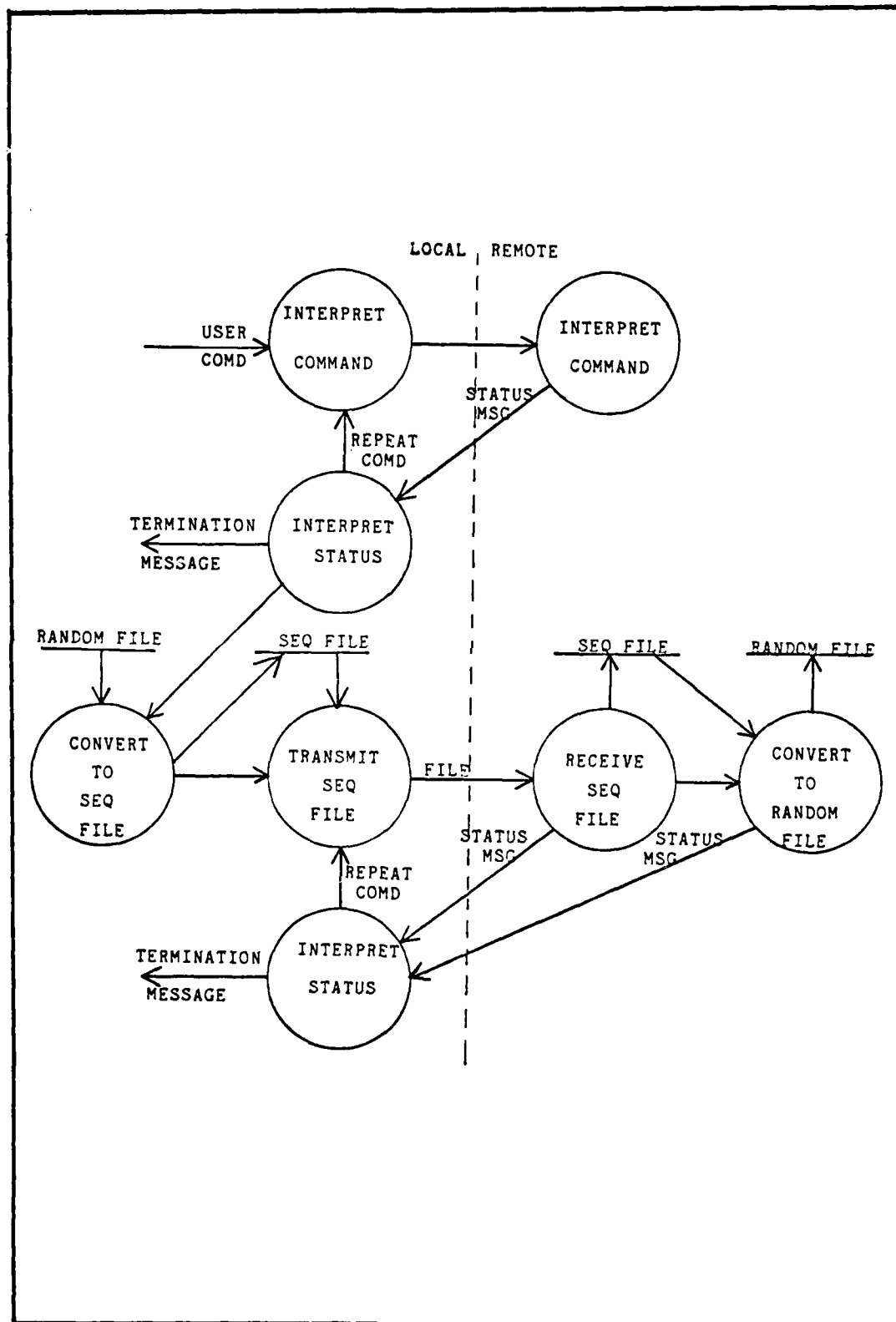


Figure 6. Transfer Program File

Transfer Message. The transfer message function allows the user to compose and transmit messages addressed to a peripheral of a remote computer. The function also allows the user to abort the message prior to transmission. As shown in figures 7 and 8, three discrete commands are available to the user. The compose command provides the user with an input/output (I/O) procedure that allows editing. If the user decides not to send a message, he can use the abort message command. If the user decides to send the message, he executes the send message command.

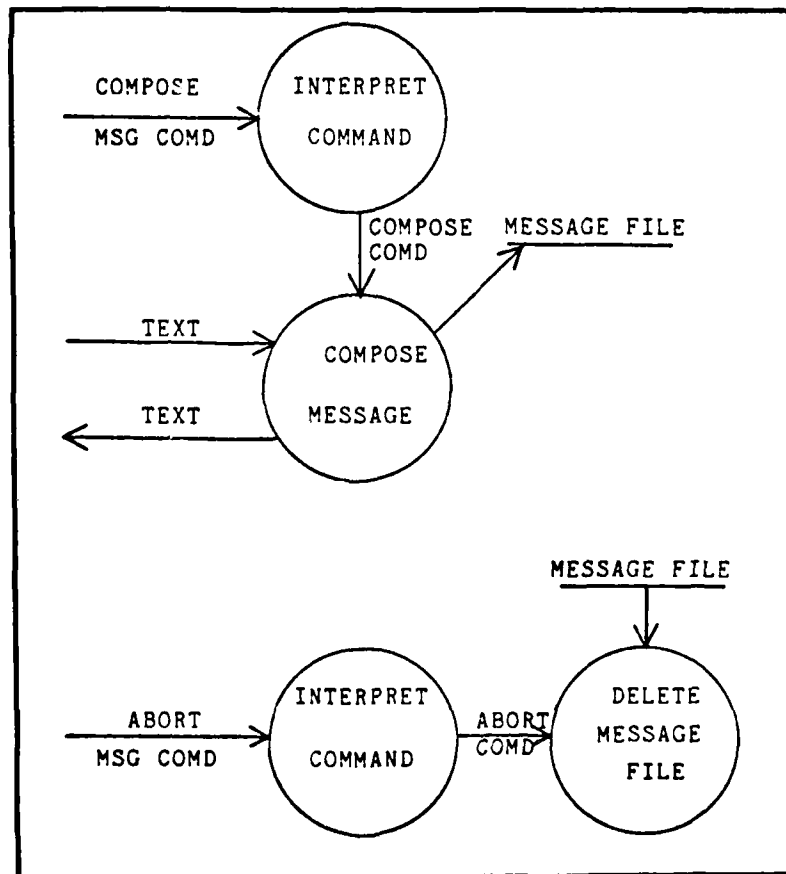


Figure 7. Compose and Abort Message

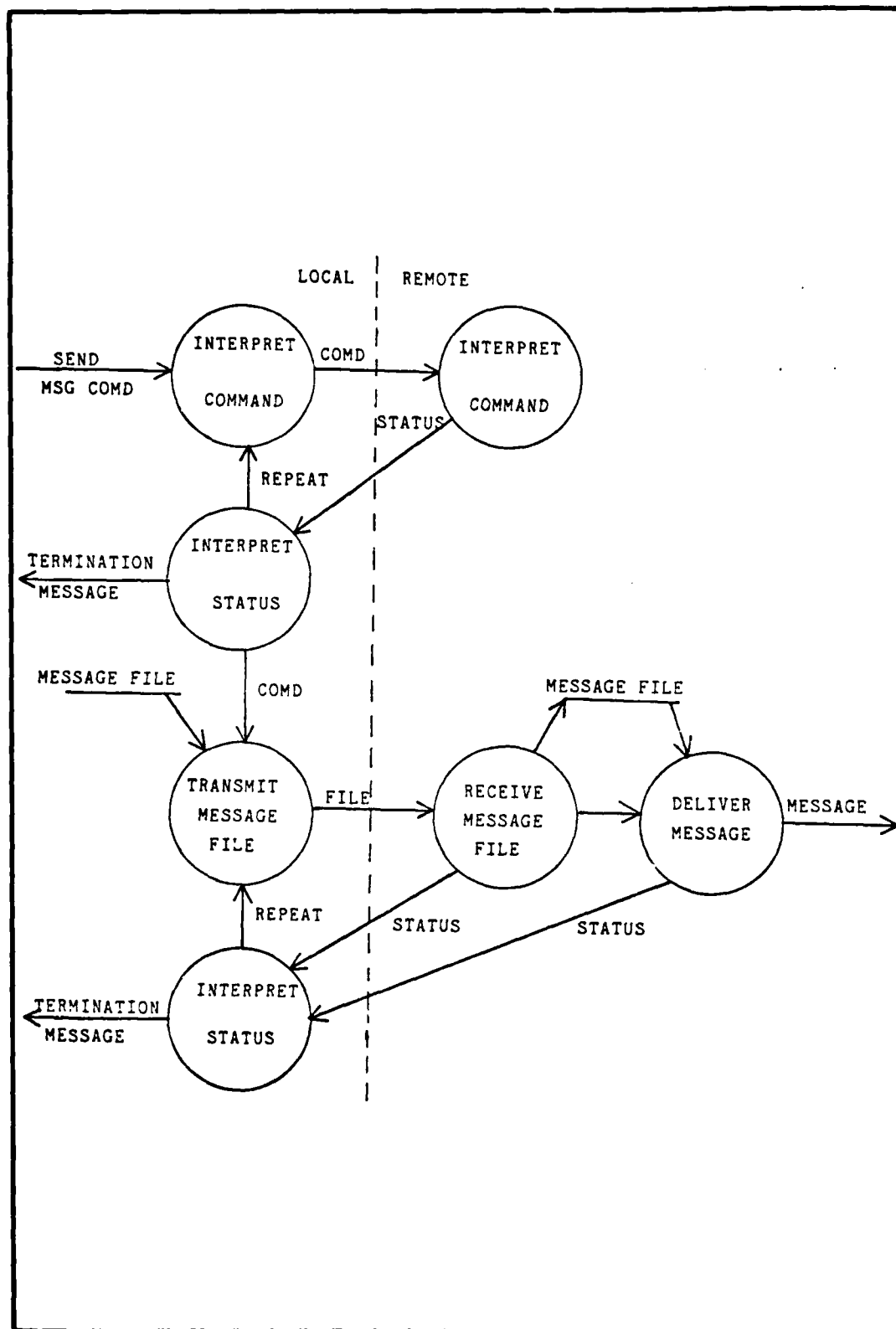


Figure 8. Send Message

Execute Remote Program. This R-S function requires relatively few processes compared to the other functions described. It involves, simply, transmitting a command which specifies the location, name, and language of the program, such as FORTRAN, Pascal, or assembly. The processes associated with Execute Remote Program are shown in figure 9.

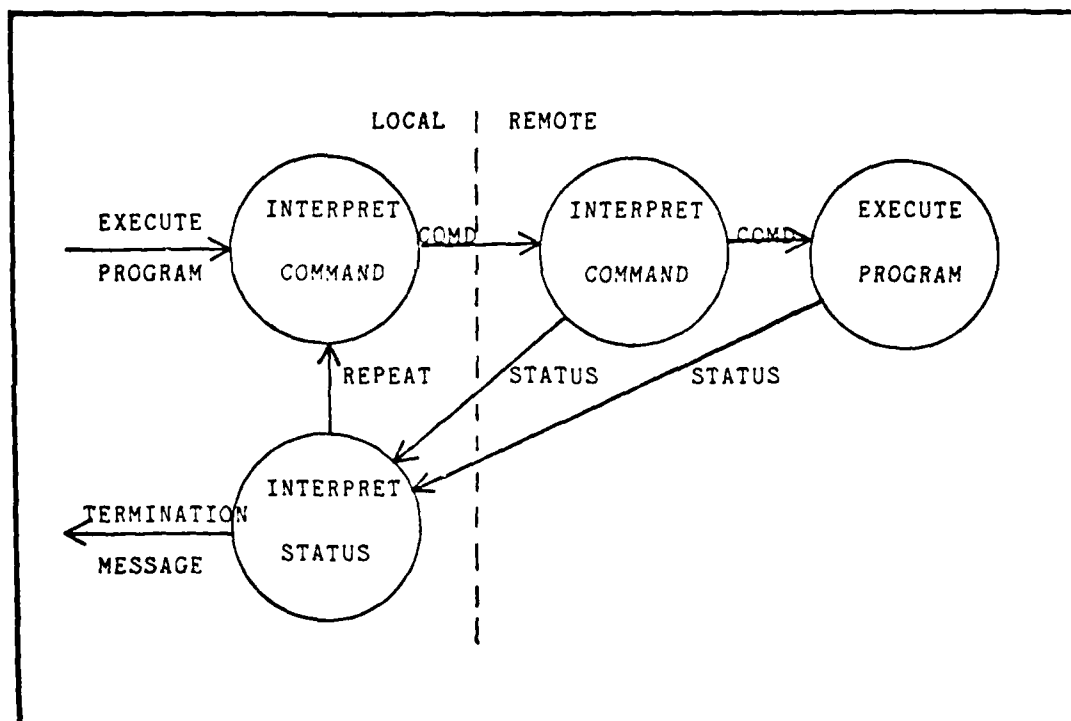


Figure 9. Execute Remote Program

Execute Remote Program Development. Program development involves several major tasks including editing, compiling, link editing, and testing. To accomplish these tasks effectively, they must remain discrete functions. Only two of the functions, compile and link edit, can not be initiated by commands defined previously. Compiling and link editing are accomplished by the Compile and Link Edit commands, respectively. The processes initiated by the commands are shown in figure 10. The DFD's are deceptively simple. Although, Compile and Link Edit are shown as single processes, they are complex processes requiring numerous parameters in addition to the basic command. The compile and link edit processes must, therefore, monitor the message and list files, respectively, to determine the progress of the requested function. Appropriate status messages must be returned to the user.

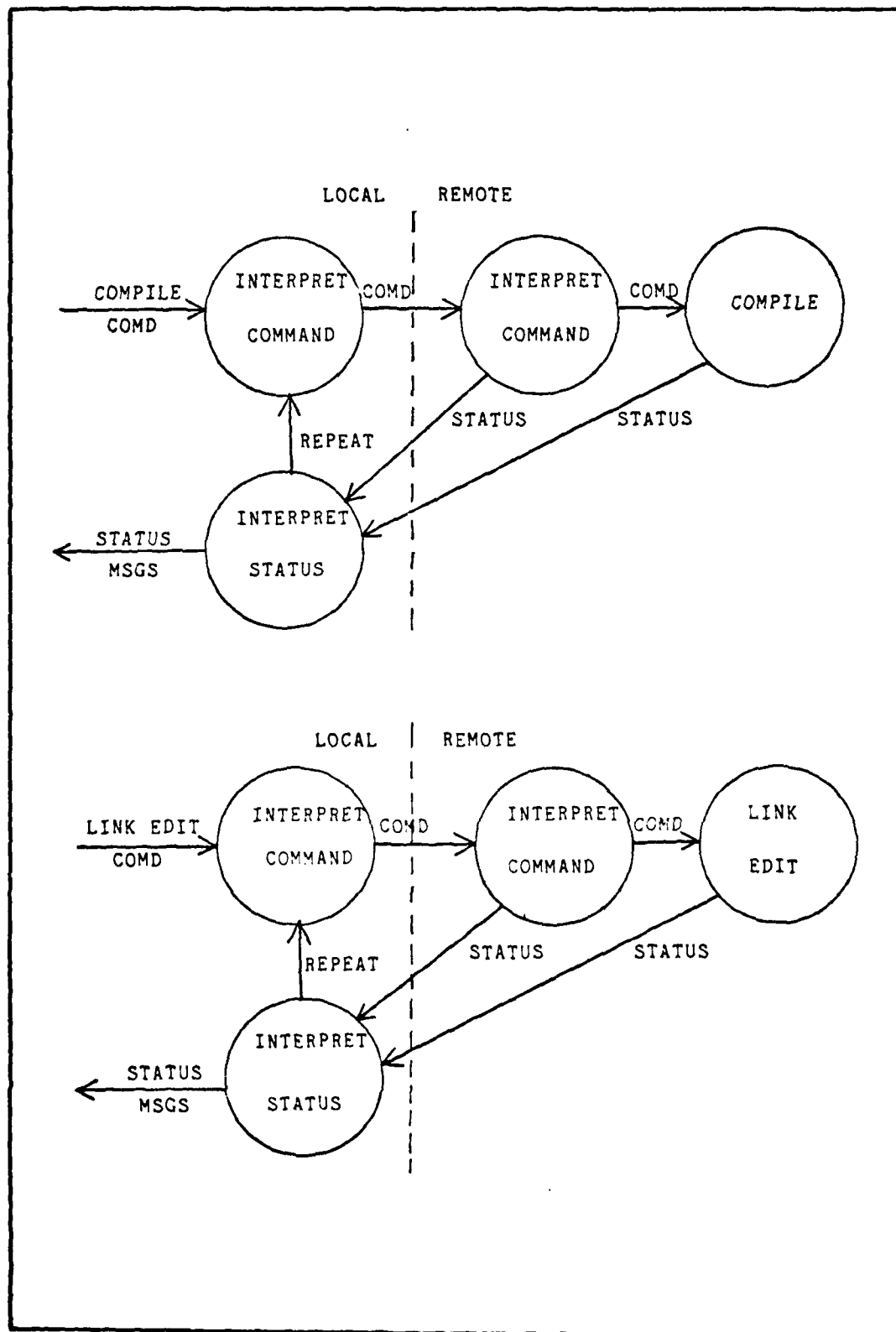


Figure 10. Compile and Link Edit

System Requirements Analysis Summary. The systems requirements analysis provides an understanding of what an R-S link must do, in general, and the processes required to accomplish each of the specific functions. From the requirements analysis it was clear that there were two important classes of processes. First, basic data communications processes are needed to facilitate a dialogue between the two computers being served by the R-S network. Second, the R-S processes themselves must be created, or, if the R-S processes already exist, they must be made accessible to the network. The following section evaluates existing AMS modules to determine which processes identified in the analysis currently exist.

Existing AMS Modules

Several existing AMS hardware and software modules were found to have capabilities that satisfy certain system requirements. Each module is described in general and its role in satisfying system requirements is identified.

Model-990 Computer. The Texas Instruments Model 990 computer is based on the TMS 9900 16 bit microprocessor. The hardware is designed to support a concept called the workspace. The workspace is a 16 word block of memory associated with each program. Each word of the workspace is rapidly accessible by the program in a manner normally associated with hardware registers. Consequently, each word

of the workspace is known as a workspace register (WR) Three hardware registers are also available; the workspace pointer (WP), Program Counter (PC), and Status Word (ST). The WP points to the first word of the programs workspace. The PC and ST monitor the address of the current instruction and the program status, respectively. When a context change is indicated, such as with a hardware interrupt or a subroutine call, the old WP, PC, and ST are stored in WR's 13,14,and15 of the newly executing program's workspace. The storing of the old pointers in the workspace registers provides an efficient means of returning to the interrupted or calling program. The overall effect of the above features of the Model 990 is rapid and efficient handling of many external interrupt sources, such as communications links. Therefore, the hardware architecture of the Model 990 is well suited to the support of computer communications, either as a host or as a node.

Peripherals. The peripherals that are associated with the Model 990 computer provide I/O and storage capabilities for the computer and the R-S link. To a large degree, peripherals are the reason that resource sharing is desirable. For instance, the SDMF has additional memory that allows it to support software development utilities that the typical site cannot accomodate. Sharing the SDMF, therefore, benefits the entire network considerably.

Peripherals that are currently available in AMS are listed below:

TI 810 Line Printer
50 MBYTE Hard Disk Drive
10 MBYTE Hard Disk Drive
Dual Floppy Disk Drive
TI 911 Vidio Display Terminal
QUME Letter Quality Printer
Tektronics Graphics Display

Communications Interface Card. The Communications Interface Card is an optional hardware module which resides in the Model 990 chassis(Ref 18). It provides an RS-232 standard interface to a data modem and supports synchrous and asynchronous data communications. In conjunction with the modem, and telephone circuit, the communications interface card implements the physical level protocols.

Vadic 3400 Data Modem. The Vadic 3400 Data Modem supports synchronous and asynchronous data communications up to 9600 BPS. The data modem provides the interface between the Communications Interface Card and the telephone circuit. The data modem contributes to the implementation of the physical level protocols.

Communications Module. The Communications Module is a program developed by the Microbase Co. to establish a data communications connection between two computers. The

program is executed by a special operating system command. A telephone number is the only parameter entered.

3780 Emulator. The Model 990 computer 3780 Emulator (Ref 20) emulates the IBM 3780 Data Communications Terminal. In other words, the 3780 Emulator implements the data communications protocols of the IBM 3780. The protocols implemented effectively span the link level to a minimal R-S capability. The link level consists of IBM's Binary Synchronous Communications (BSC) protocol. The network and transport layer functions are not evident since the Emulator supports a simple link between two computers without routing or multiplexing. The presentation layer function that is evident in the Emulator is the process of transforming the TI sequential file format into an IBM sequential file format and vice versa. a liaison, or session, is established between the two computers when the Emulator of one is commanded to ANS (answer) and the other is commanded to CALL. The Emulator can be considered to have a minimal R-S layer since it is capable of transferring sequential files, an attribute associated with the R-S layer. The Emulator can also execute a remote program if it is in the operating system program file. By interfacing to the Emulator, a program is provided with all of the data communications functions associated with a minimal R-S level and below. The Emulator provides all the basic data

communications capabilities needed by an R-S process.

Operating the Emulator involves entering commands and monitoring log output. Commands may be entered from any one of three sources; a terminal, a command file, or an intertask message queue. The desired mode is designated upon activation. Similarly, the log output may go to one of three destinations; a terminal, a log file, or an intertask message queue. In its role as a R-S network link module, the Emulator interfaces to the R-S process via two intertask message queues.

DX-10 Operating System. The operating system associated with the Model 990 computer used in this investigation was the DX-10, release 3.3, operating system(Ref 18). It is an interactively oriented operating system, but it also supports a batch mode environment. DX-10 is a multi-terminal system capable of making each of several users appear to have exclusive control of the system. Program segmentation is supported such that each program may have up to one task and two procedure segments. If procedure segments are reentrant, or pure, they can be shared with other tasks. Each terminal can have two tasks active simultaneously; a foreground, or interactive, task and a background, or noninteractive task. Higher order languages supported include FORTRAN, Cobol, RPGII, Pascal, and both scientific and business Basic. A text editor, link

editor, and debugging facility are included with the program development capabilities. Other utility programs include a data base management package and a Sort/Merge package.

The operating system proved to be the single most important asset available for satisfying the R-S link requirements. Since an operating system is the controller of each computer's resources, the only way to access those resources is through the operating system. Most of the resources exist external to the operating system, with the operating system controlling access to them. However, the operating system may also be the direct supplier of the resource. Figure 11 shows the basic relationship of the operating system to the other R-S link resources.

Operating System Interfaces

Before a system design could be accomplished, the interfaces to the operating system had to be understood. The following subsections describe those interfaces.

Operator/Operating System. The operating system consists of many independent tasks which are controlled by a supervisory task called the System Command Interpreter (SCI). The SCI is primarily intended to be an interactive interface that accepts commands from an operator. However, upon receipt of an execute batch (XB) command from the operator, a copy of SCI is executed in the background that obtains commands from a "batch file" designated by the operator.

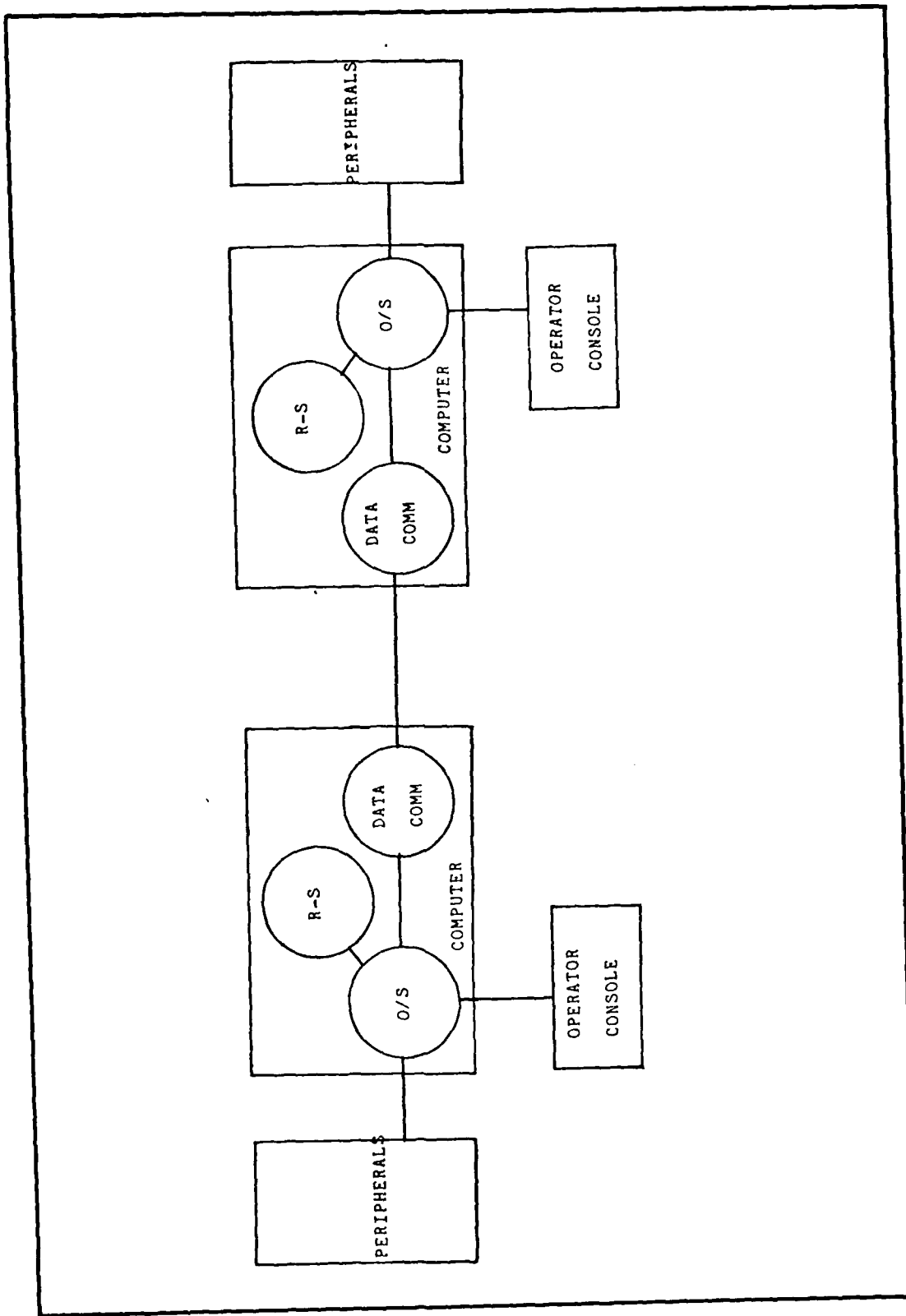


Figure 11. Operating System relationships

The SCI language consists of primitives and commands. A primitive is processed directly by the SCI but commands are interpreted by SCI by reading a sequential file called a Command Procedure (PROC) which can contain primitives and commands. Commands can be nested in other commands as indicated by the fact that a PROC may contain commands. Primitives may be entered directly at a terminal, but they are primarily intended for use in PROCs. A primitive consists of a period followed by a mnemonic. The primitive is followed by mandatory and optional parameters. A typical primitive is .BID which causes a task to be executed in the foreground mode. A typical command is Execute Task (XT) which is similar to .BID and, in fact, contains a .BID. The XT command, however, is operator oriented and aids the operator by prompting for parameters. Primitives such as .IF, .ELSE, and .ENDIF exist, allowing program-like decisions in a PROC.

A Command Procedure (PROC) usually causes the execution of a program which is permanently associated with a particular command. A program which is permanently associated with a command is called a Command Processor. A Command Processor uses special operating system subroutines to access parameters in the Terminal Communications Area (TCA). The TCA is a special portion of memory associated with each interactive terminal. The parameters are placed

in the TCA by SCI as it interprets the command PROC.

In summary, the Operator/Operating System interface provides a versatile, uniform interface for the operator. The SCI is commanded by a simple, operator entered mnemonic that causes the SCI to prompt the operator for the necessary parameters. SCI then executes the appropriate O/S programs and command processors. Commands may be added to the SCI repertoire by the user. R-S Network Commands defined by this investigation were defined as SCI commands. The software portion of the R-S process developed by this investigation acts as a command processor and is executed by each of the network commands.

Applications Program/Operating System. An applications program can communicate with the operating system with supervisory calls (SVC). There are more than 200 SVCs. Most of the SVCs are included in three classes; file and I/O calls, Program Control calls, and memory control calls. Supervisory calls are provided expressly to enable an applications program to command the operating system. For most programs, the SVCs provided are adequate. However, to satisfy the R-S link requirements a more powerful interface was needed in addition to the SVCs. The special interface developed by this investigation is described under "Applications Program/SCI".

Operator/Applications Program. The primary means of

implementing communications between the operator and an applications program is by including I/O supervisory calls (SVC) in the program that communicates with the terminal. Additionally, there is a degree of communications prior to the execution of the program, if the program is defined as a command processor. If the program is a command processor, the operator is prompted for parameters needed by the program and the operator entered parameters are subsequently fetched by the program from the Terminal Communications Area (TCA).

Applications Program/Applications Program. An application program may communicate directly with another applications program by using the Intertask Communications (ITC) supervisory call. The ITC SVC provides a queue into which a program may place messages, each containing up to 92 Characters. The queues are usually established with a 1000 character capacity at system generation, but they can be smaller or larger. The messages may be fetched by another applications program on a first in first out (FIFO) basis.

Applications Program/SCI. The SVCs provide an adequate interface to the operating system for most applications. However, the system requirements analysis for the R-S link revealed several basic processes which were contained in the O/S but were not accessible with SVCs. The required processes could, conceivably, have been duplicated.

However, the processes in question were not trivial, making it impractical to redevelop them. For instance, the process to transform a random file into a sequential file is an inherent capability of the O/S. Another example is the process of compiling. Obviously, a suitable interface between the R-S process and the O/S had to be found or developed.

A thorough study of the DX-10 Operating System Manual (Ref 18) did not reveal a means of achieving the desired interface. However, an operating system subroutine was discovered through a supplementary source (Ref 5). The subroutine had the essential capabilities to effect the interface. The assembly language routine found is named S\$BIDT. It was designed to allow an applications assembly language routine to execute a program that fetches its parameters from the TCA. The applications assembly program calls S\$BIDT and passes parameters to S\$BIDT according to a prescribed format. Subsequently, S\$BIDT places the parameters in the TCA and executes the program that fetches the parameters from the TCA.

The operation of S\$BIDT is important because many O/S programs, including SCI, use the TCA to acquire their parameters. Some SCI commands result in the SCI executing other independent O/S programs. For those commands, the SCI could be emulated by executing those programs through

S\$BIDT. However, that does not result in a good general solution to the interface problem because many SCI commands result in SCI calling an overlay of itself. Since there is no way to execute an overlay independently of its root task, it was not possible to emulate the SCI for all commands. The solution lay in emulating the Execute Batch (XB) command and designating a batch file that contained the desired commands.

Fortunately, the XB command could be emulated through the use of S\$BIDT. The XB command prompts the SCI to read a PROC which contains a .QBID primitive which instructs the SCI to execute a specified task in the background mode. The XB PROC contains the .QBID followed by three parameters; the task ID of SCI, a batch file pathname, and a batch list pathname. To emulate the XB command, a background program must execute S\$BIDT and pass the necessary parameters. S\$BIDT places the two file parameters in the TCA and executes a background copy of SCI. Upon execution, SCI fetches the parameters and executes the primitives and commands in the designated batch file.

Although an applications program could emulate the SCI in some instances by executing the appropriate O/S routines with the S\$BIDT interface, most SCI commands involve an overlay of the SCI. Therefore, it would usually be necessary to emulate the XB command by executing SCI through

S\$BIDT. As a result of these considerations, it was decided that the S\$BIDT/SCI interface should be the standard applications program/operating system interface when an SVC is not available. This simplifies the programming of the R-S process by eliminating the possibility of interfacing with many O/S programs with varied parameter requirements. A more detailed description of the S\$BIDT/SCI interface is contained in the Appendix.

System Design

This section describes the combining of new and existing modules into a system that fulfills the system requirements. The existing modules described in the preceding section possess most of the basic capabilities needed to satisfy the R-S link requirements. However, if the modules were allowed to remain as independent processes in each link computer, the required R-S functions could be realized only by having an operator at each end of the link. The operators would have to communicate via telephone and execute numerous commands at the proper times to achieve the necessary synchronism. To automate the functions, a coordinating process is required at each end. The coordinating process is the R-S process at each end of the link. The two R-S processes effectively combine the many processes in each computer into two cooperating network processes. The interfaces described earlier in this chapter

enable the R-S, applications process to accomplish that goal.

The system design accomplished by this investigation is shown in figure 12. The R-S process has been divided into two processes, a foreground executable process known as Foreground Network (FORENET) and a background executable process known as Background Network (BACKNET). The Data Communications process exists as two processes, the 3780 Emulator and the Microbase Communications program. The peripherals that exist at each end of the link are also illustrated and the operating system is shown connecting all of the resources.

The operation of the system can best be described from the perspective of the R-S processes, FORENET and BACKNET. The typical R-S function is performed under the supervision of two R-S processes, a local FORENET and a remote BACKNET. FORENET and BACKNET cooperate to accomplish the many functions implied by the R-S command entered by an operator or a user process. Figure 13 is a simplified view of the R-S link with only R-S processes and the command sources visible. The data communications processes are transparent to the R-S processes.

The execution of an R-S function begins when an R-S SCI command is entered. The SCI interprets the command by reading primitives and commands in the PROC associated with

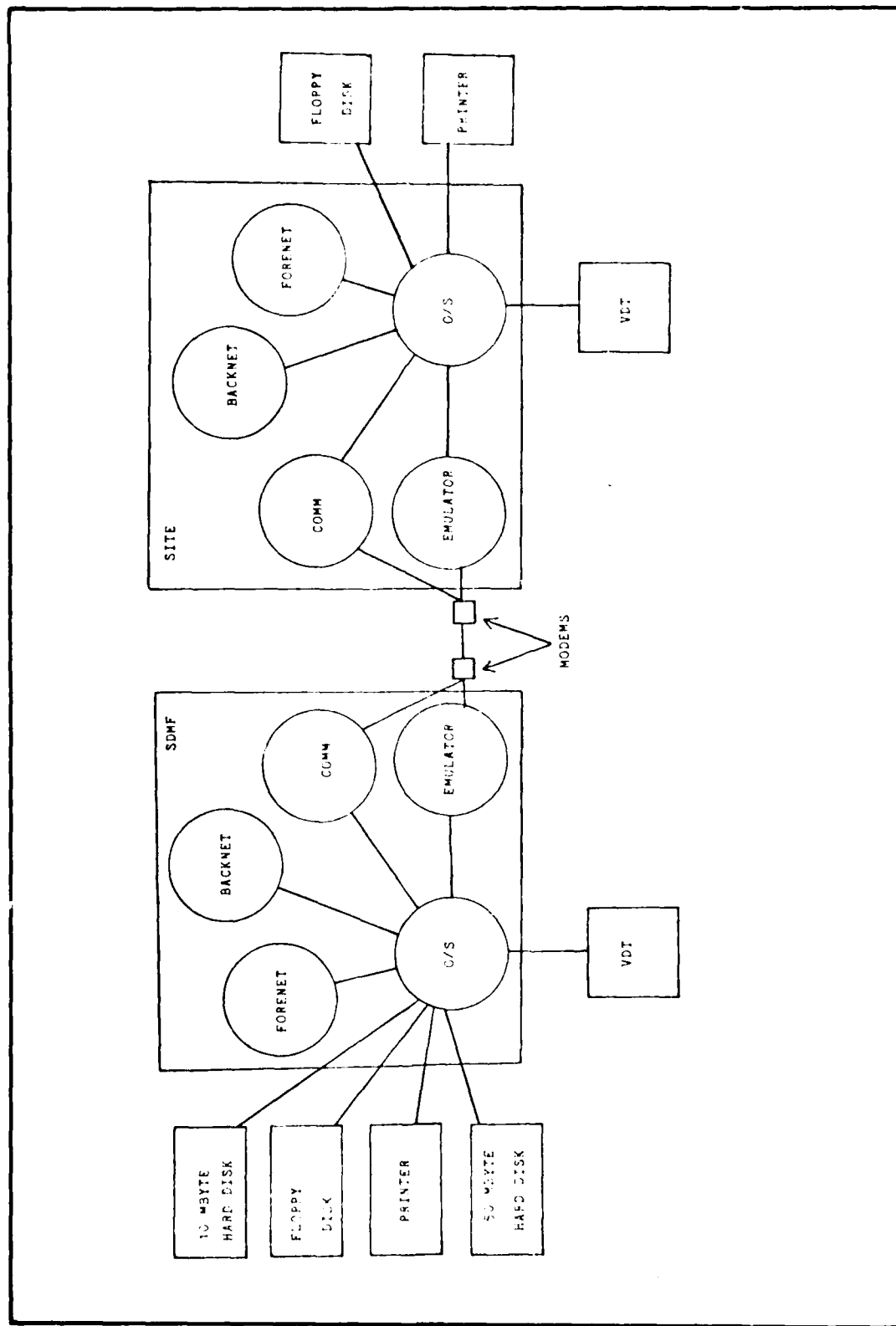


Figure 12. System Design

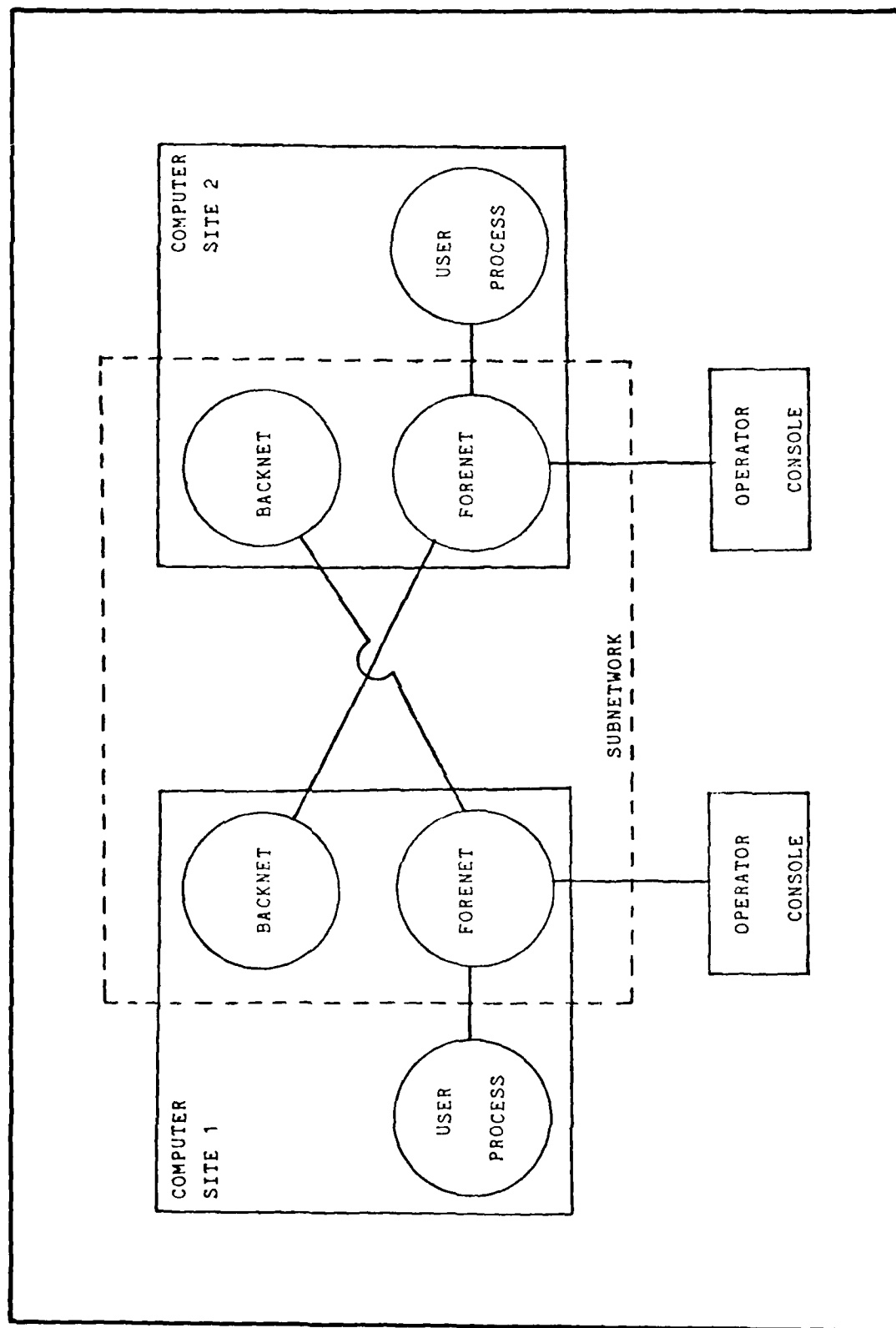


Figure 13. FORENET and BACKNET R-S Processes

the command. The typical PROC will cause SCI to acquire parameters associated with the command and execute the local FORENET in the foreground mode. The first action FORENET takes is to check if the local BACKNET is active. If the local BACKNET is active, FORENET informs the user that the link is busy. Otherwise, the local FORENET executes the remote BACKNET and communicates the command type. The interchange continues between FORENET and BACKNET until the R-S function is completed.

There are several reasons for the selection of the foreground and background modes associated with FORENET and BACKNET, respectively. BACKNET is always the remote process and sometimes the local process. In either case, BACKNET must execute SCI in the batch mode through the S\$BIDT/SCI interface. Since the S\$BIDT/SCI interface can only be used by a process in the background mode, BACKNET must always be executed in the background mode. Since FORENET is intended to be an interactive process, it is convenient for it to be in the foreground mode where O/S interactive features are most prevalent. Additionally, it is convenient for FORENET to be executed in the foreground so that it can be active at the same time as BACKNET. Even if the link is busy because an operator at the other end has activated it, the local operator will be able to initiate a command, and FORENET will determine that the link is busy by checking the status

of BACKNET and notify the operator appropriately. If an attempt was made to execute FORENET in the background mode at the same time as BACKNET, the operator would receive an ambiguous O/S error that an unspecified task was already active in the background mode.

This section has described the system design of the R-S link. The system design defines the relationships of the R-S processes to the existing AMS modules. The system requirements are satisfied through the ability of the R-S processes to interpret R-S network commands and to coordinate the actions of several independent, subordinate modules. The apparent effect is that the command is performed by only two processes, a remote and a local process.

Network Commands

With an understanding of the system design it was possible to define a minimum set of commands. Most of the commands defined were directly related to the R-S functions defined in the system requirements. However, others were needed for initialization and control purposes.

Also, a batch command was defined to allow a general method of accessing remote computer resources. It is obviously not a simple or highly automated command, but it does provide a means of accessing any resource of a remote computer if the operator is skilled in the SCI language.

The following network commands were defined.

NI -- Network Initialize
NLON -- Network Log On
NTF -- Network transfer file
NMC -- Network Message Compose
NMS -- Network Message Send
NMA -- Network Message Abort
NTPF -- Network Transfer Program File
NXT -- Network Execute Task
NCOMP -- Network Compile
NLINK -- Network Link Edit
NXB -- Network Execute Batch
NLOFF -- Network Log Off
NQ -- Network Quit

The set of commands represents a complete set in the sense that they result in all of the required R-S functions being performed.

Summary

This chapter has analyzed the R-S Network Link system requirements and described a system design that satisfies those requirements. The system requirements were analyzed to determine the processes needed to satisfy the requirements and Data Flow Diagrams were used to illustrate the processes. Existing AMS hardware and software modules that partially satisfy the system requirements were

described. A system design was described that consists of existing AMS modules and the newly defined R-S processes. Based on the system design and the system requirements a set of network commands were defined. The next chapter describes the analysis of the R-S network commands to determine what processes must be implemented in the R-S software to satisfy the software requirements.

III. Software Requirements Analysis

This chapter describes the analysis of the Resource-Sharing (R-S) software requirements. The analysis determines the processes that must be implemented in the R-S software by analyzing what the software must do in response to the network commands. Data flow diagrams are shown for each command to illustrate the processes and data needed for the command. The processes are grouped as they would be geographically to enhance visualization of the protocols described for each command.

General Information

There are several features that appear in all or most of the network command data flow diagrams (DFD). The features include conventions of notation, methods of passing command parameters across the R-S link, and communications techniques between R-S processes at opposite ends of the link.

As shown in figure 12, Chapter 2, the major processes at each end of the link include two R-S processes, the 3780 Emulator, and the operating system. The only operating system process to appear in the command analyses was the System Command Interpreter (SCI). To differentiate between local and remote processes, the process abbreviation is suffixed with a 1 or a 2. Therefore, the local processes are

FORENET 1 (FN1), BACKNET 1 (BN1), 3780 Emulator 1 (EM1), and System Command Interpreter 1 (SCI1). Similarly, the remote processes are identified by FN2, BN2, EM2, and SCI2. Standard SCI commands are referred to as SCI commands, but network commands which are also implemented as SCI commands are referred to as network commands.

Figure 14 shows the configuration of processes for typical network command. The circle representing each process is identified by the process name and the abbreviation of the parent process. There two differently shaped arrows indicating connectivity between R-S processes and the log and command ports of the data communications (Emulator) processes. The "filled" arrows designate the common connection points for arrows of like shape. Each DFD illustrates the SCI interface to an operator. The applications process interface to R-S processes is described separately at the end of this section.

The first active process is SCI1 Interpret Command. SCI interprets a network command by reading the Command Procedure (PROC) associated with the command. The PROC contains SCI commands and primitives that instruct the SCI to prompt the operator for network command related parameters (PARMS), to place those parameters in the Terminal Communications Area (TCA), and to execute FN1.

Upon execution, FN1 indirectly executes BN2 by

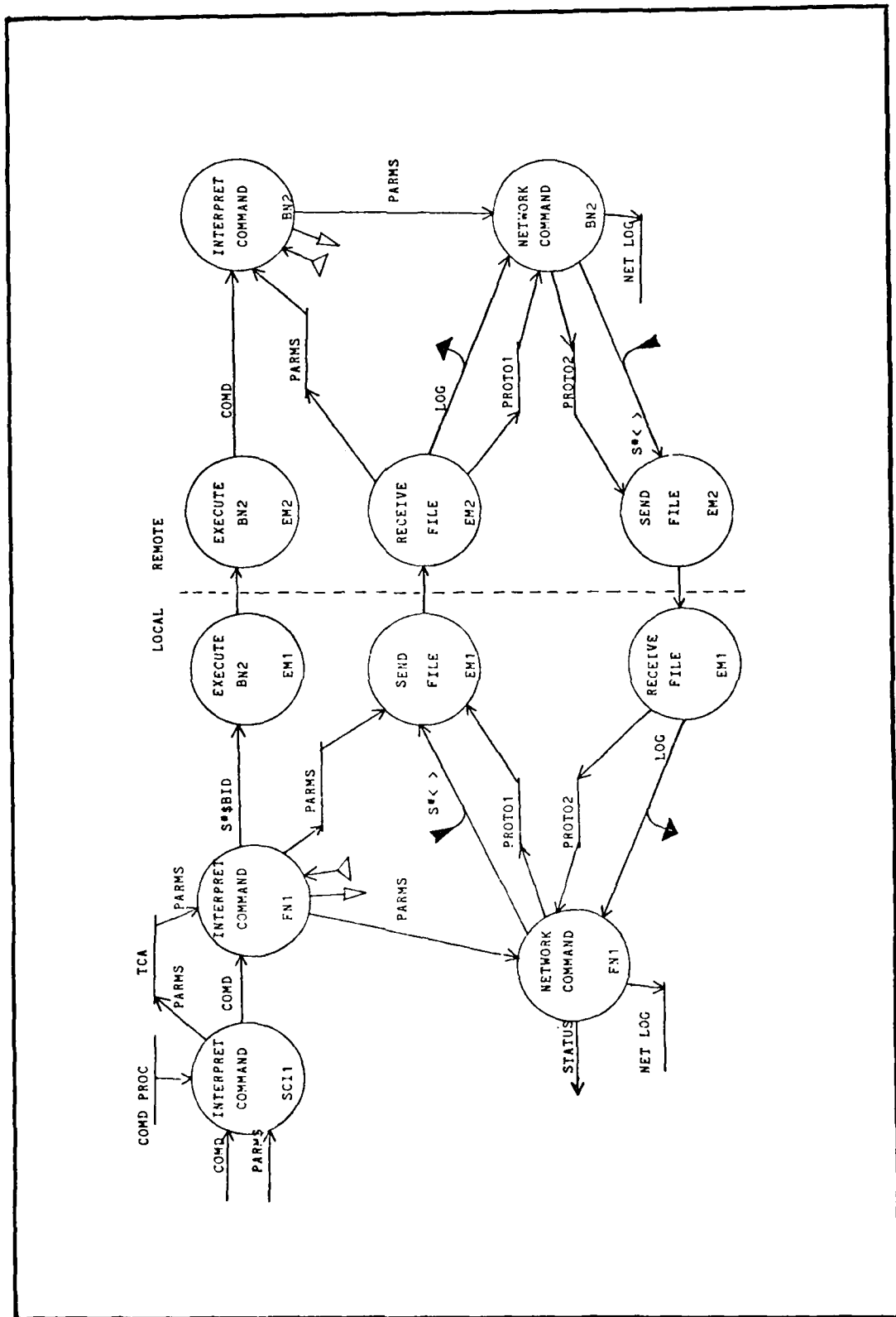


Figure 14. Typical Command

commanding the Emulator with a S*\$BID*< > Emulator command. The S*\$BID*< > command causes the execution of a remote task that has been installed on the S\$PROGA system program file of the remote Model 990 computer. The angle brackets represent required and optional parameters that accompany an actual command. Typical parameters include the name of the remote task and a 32 bit "code" word. In the analysis of the network commands the code word is used to transfer a network command mnemonic. A detailed description of the 3780 Emulator commands is contained in Ref 20.

Once a complementary pair of R-S processes have been executed, they communicate by using the Protocol 1 and Protocol 2 files. For example, after the BN2 Interpret Command process has been activated, it acknowledges activation and recognition of the network command by sending a predetermined message in the Protocol 2 file. Upon receipt of the acknowledgement, the FN1 Interpret Command process passes the command parameters to BN2 with the PARMS file. The PARMS file is a special file used only for the transfer of command parameters.

The BN2 Interpret Command Process decides which network command process is to be activated by examining the network command and its parameters. There is at least one unique process associated with each command. When BN2 Interpret Command process decides which command process to activate,

the command parameters are passed to that process. The transfer of parameters also implies transfer of control.

Upon activation by the BN2 Interpret Command process, the network command processor establishes communications with the FN1 network command processor by exchanging protocol messages via the Protocol 1 and Protocol 2 files. A process transfers a file, such as Protocol 2, by placing a command in the Emulator Command Queue of the form S*< >. The source and destination file pathnames replace the angle brackets in the actual command. A process detects the arrival of a file, such as Protocol 2, by reading the Emulator Log Queue messages. An Emulator Receive Pathname (RPN) message in the log Queue signals the arrival of a specific file. The Emulator Log Queue can also contain other status and error messages.

The network command processes also maintain communications with their complementary processes at the opposite end of the link via the Protocol 1 and Protocol 2 files. The command processes must communicate frequently to synchronize their activities. The local command processes send status and error messages to the operator or user process, and all command processes write messages to the Network Log file. Additionally, most command processes read and write to other files to accomplish the specific objectives of a network command. Finally, the command

processes must exchange protocol messages to determine if all objectives of a network command have been achieved and terminate themselves.

A network user process can command the R-S processes by interfacing with FN1 through the S\$BIDT operating system routine as described in the previous chapter. Figure 15 shows the processes that facilitate the interface. From the perspective of the Interpret Command process, the interface is identical to the SCI1 interface because S\$BIDT is capable of performing all of the functions that SCI1 was commanded to perform by the PROC. Specifically, S\$BIDT places the parameters in the TCA, executes FN1, and passes the command mnemonic to FN1. Conversely, the PROC used to facilitate the operator interface must only contain the SCI commands required to acquire parameters and execute FN1, if the

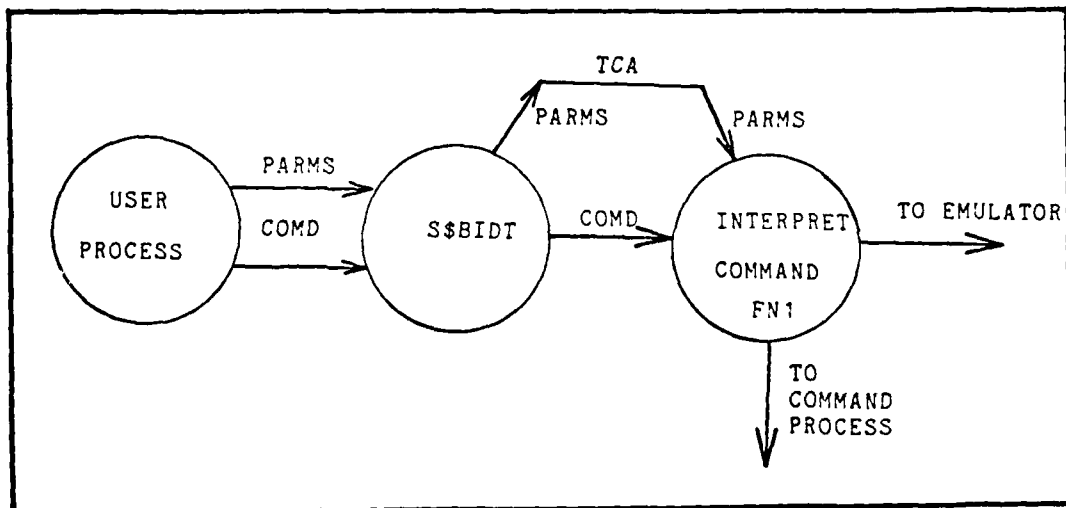


Figure 15. User Process/ R-S Process Interface

command is intended to be executable by a user process also. The reason for the limitation on PROC commands is that a foreground process cannot emulate the Execute Batch (XB) SCI command. XB can only be emulated by a background process as described in the previous chapter.

The typical command described in this section is applicable to all of the network commands except Network Initialize (NI), Network Quit (NQ), and Network Transfer Program File (NTPF). NI and NQ involve local processes only and, therefore, does not require a liaison between processes across the link as described above. NTPF is also an exception because both the local and remote R-S processes need access to the SCI through the S\$BIDT/SCI interface. The result is a more complicated initialization procedure to establish a liaison between both the local and remote BACKNET processes. A description of each network command analysis is contained in the following sections.

Network Initialize (NI)

Network Initialize (NI) executes and initializes the 3780 Emulator, enabling the local AMS site to receive calls from a remote site. The processes necessary to realize the NI network command are shown in figure 16.

The first active process is SCI1 Interpret Command. SCI interprets the NI command by reading the NI PROC. The first action of SCI would normally be to acquire parameters,

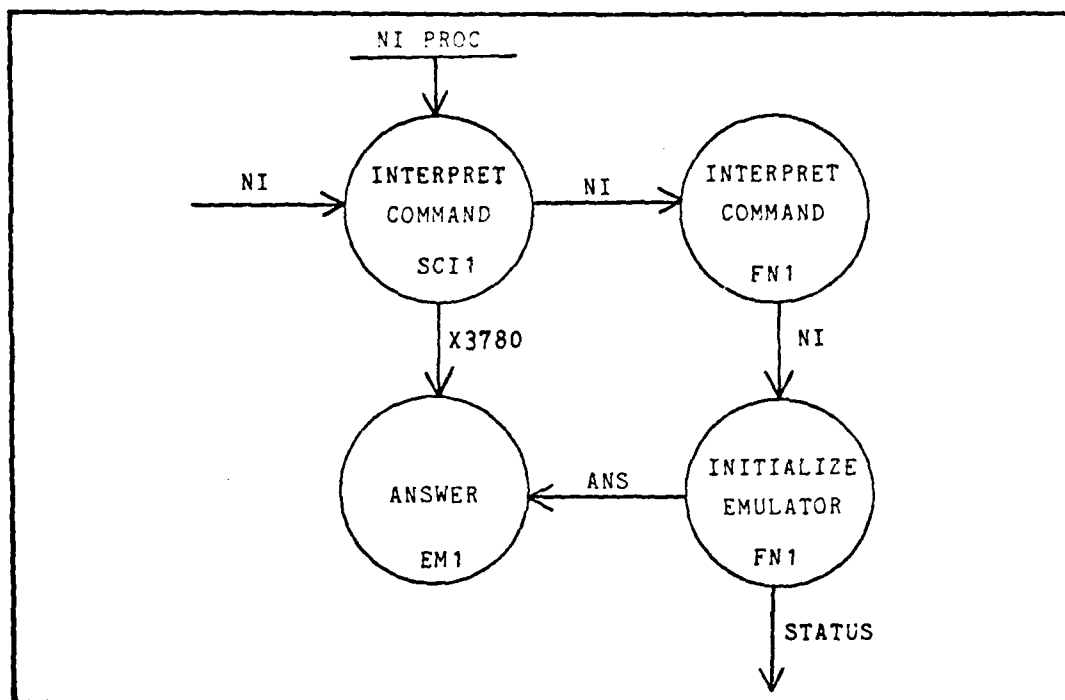


Figure 16. Network Initialize (NI)

but NI does not require any parameters. The first SCI command in the PROC is X3780 which needs two parameters which are the same each time NI is executed. The X3780 command and its two parameters result in the execution of the Emulator and initialization of the Emulators command and log modes. To enable the Emulator to receive commands for applications processes such as FORENET and BACKNET, the Emulator must be initialized to receive commands from an Intertask Message Queue. To further satisfy the Emulator initialization requirements, the Network Log file is specified as the Emulator log message destination. However, each time either FORENET or BACKNET are activated, the Emulator is commanded to place log messages in specified

Intertask Message Queue (IMQ). The last command in the PROC is a .BID primitive that executes FN1.

FN1 Interpret Command activates the Initialize Emulator process. Initialize Emulator places an Emulator ANS (answer) command in the Emulator command IMQ. After receiving verification from the log queue that the Emulator is in the answer mode, Initialize Emulator logs the status of its activity in the Network Log, send a status message to the user, and terminates. The Emulator remains active and in the answer mode indefinitely.

Network Log On (NLON)

The primary objective of the Network Log On (NLON) command is to establish communications between the Emulators of two sites. NLON also logs the identity of the user and the time the communications session began in the Network Log file at each site. Figure 17 shows the processes needed to realize the NLON command.

The SCI1 Interpret Command process acquires the necessary parameters and executes the Microbase Communications (COMM) program. The COMM program establishes the physical and electrical connection between the Emulators of the sites designated by the user parameters. After the COMM program terminates, SCI1 executes FN1.

FN1 begins the procedure of establishing a liaison between the R-S processes of the two sites by executing

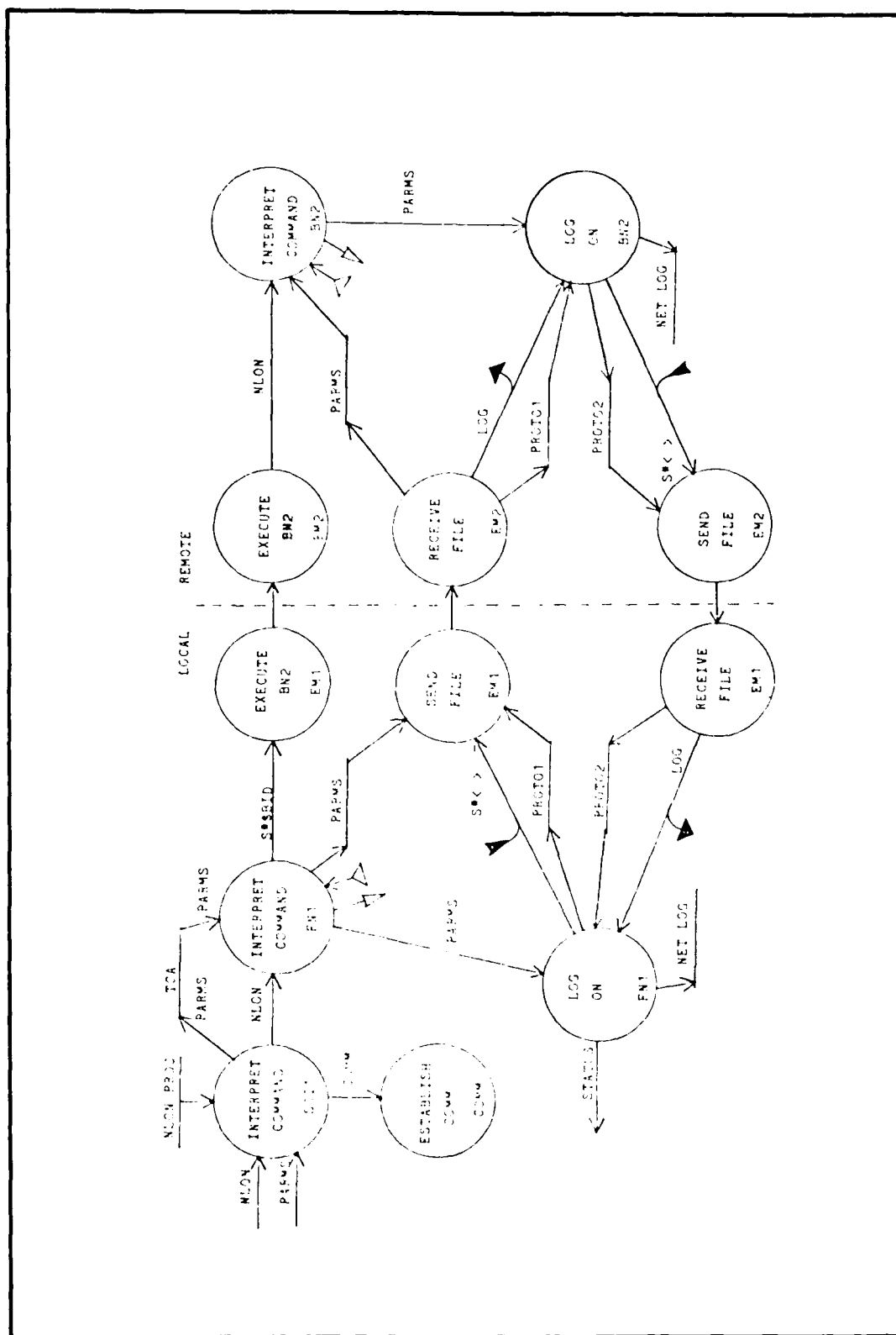


Figure 17. Network Log On (NLON)

BN2. FN1 executes BN2 via the Emulator with a S*\$BID*NLON Emulator command. BN2 acknowledges the receipt of a NLON command and FN1 continues the dialogue by transferring the PARMS file. The last action of the complementary Interpret Command processes is to pass parameters, and thereby control, to their respective Log On processes.

The Log On processes establish communications via the Protocol 1 and Protocol 2 files. Having established communications, they log the necessary information in their Network Log files. The local Log On processes sends a status message to the user and waits for a completion message from the remote Log On process. After receiving the completion message, the local Log On process returns a completion message. Finally, both processes terminate. The Emulators remain active and connected, waiting to provide communications associated with another network command.

Network Transfer File (NTF)

The Network Transfer File Command causes a specified sequential file to be transferred from the local site to the remote site, or vice versa. The specified file may be any sequential file of either computer. The processes necessary to realize the NTF command are shown in figure 18.

The execution of NTF begins with the usual preliminary actions by the Interpret Command processes of SCI1, FN1, and BN2. SCI acquires parameters, executes FN1 and passes

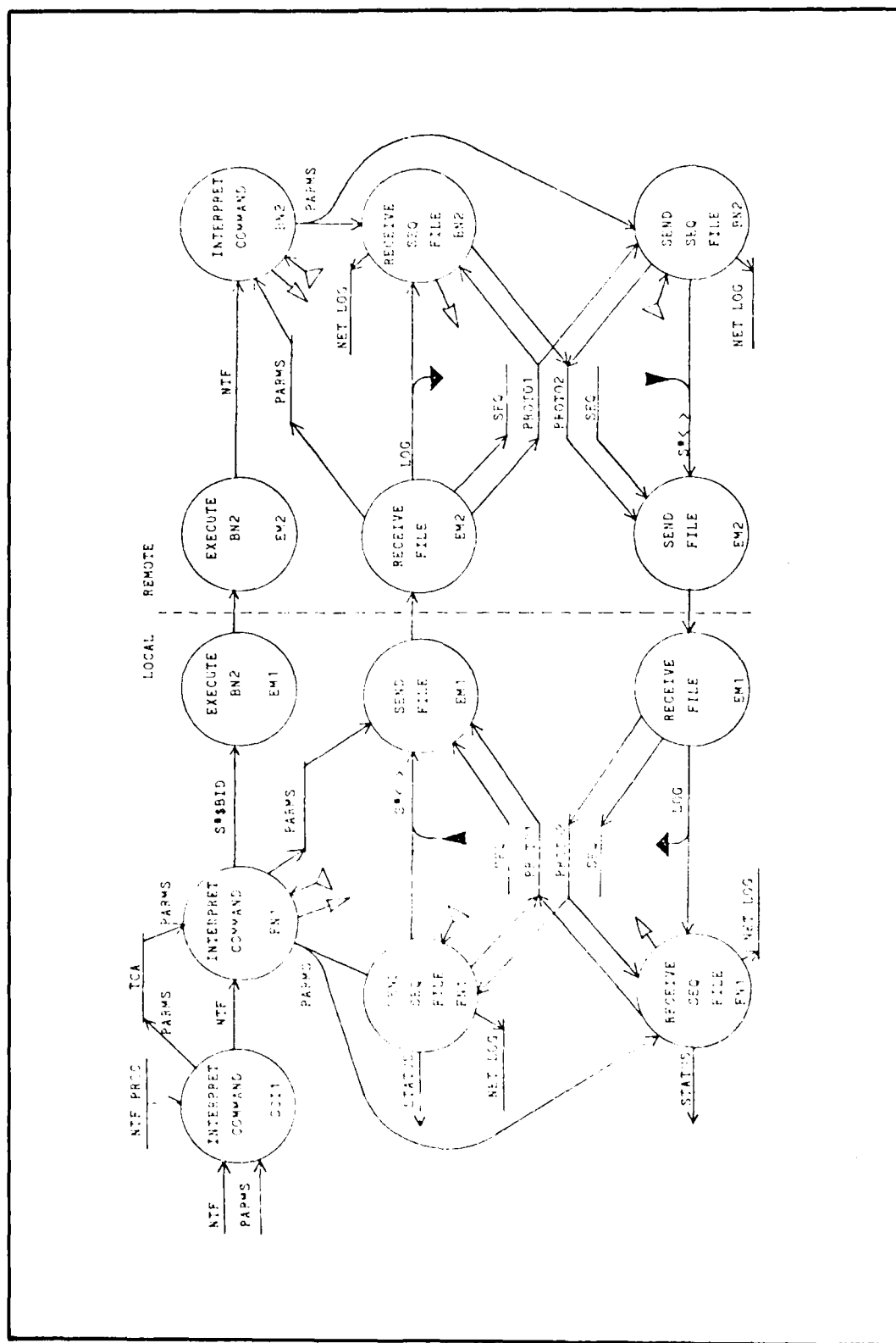


Figure 18. Network Transfer File (NTF)

parameters to FN1. FN1 executes BN2 and passes parameters to BN2.

The unique actions associated with the execution of NTF begin with the passing of control from FN1 and BN2 to one of two R-S process pairs. If the NTF command has specified that a local file be transferred to the remote end, FN1 passes NTF parameters to the process Send Sequential File and BN2 passes NTF parameters to the process Receive Sequential File. If a remote file has been specified for transfer to the local end, parameters are passed to the Receive Sequential File and Send Sequential File processes at the local and remote ends, respectively.

If the sequential file is being transferred between local and remote ends, the transactions between Send Sequential File 1 (SSF1) and Receive Sequential File 2 (RSF2) proceed as follows. SSF1 commands the Emulator with a S*< > command designating the file to be transferred. RSF2 waits for a message from the Log Queue indicating that a file with the specified destination pathname has been received. During the course of these transactions the processes at each end make entries in the Network Log file. As a minimum, the source and destination pathnames are logged. Any errors reported by the Emulator are also logged. If SSF1 receives an acknowledgement of receipt from RSF2 or detects an error from the Emulator, SSF1 sends a

termination message to RSF2 and a status message to the user, and terminates.

If the transfer is specified to be from the remote to the local end, the local Receive Sequential File and remote Transmit Sequential File processes are activated. The transactions between the processes are the same as those described above, with the exception that the Receive Sequential File process sends status to the user instead of the Transmit Sequential File process.

Network Message Commands

The network message commands provide the operator with the capability of composing a message and, subsequently, either sending or aborting the message. The Three message commands are Network Message Compose (NMC), Network Message Send (NMS), and Network Message Abort (NMA). The network message commands are entered by an operator only. The NTF command invokes an applications process with the capability to send a message to a remote applications process. The DFDs for the network message commands are shown in figure 19.

Network Message Compose (NMC). The NMC command prompts the operator for the message destination and causes SCI1 to enter the editor mode. Both of these NMC actions can be accomplished with SCI commands in the NMC PROC. The acquired parameters are equated to operating system synonyms

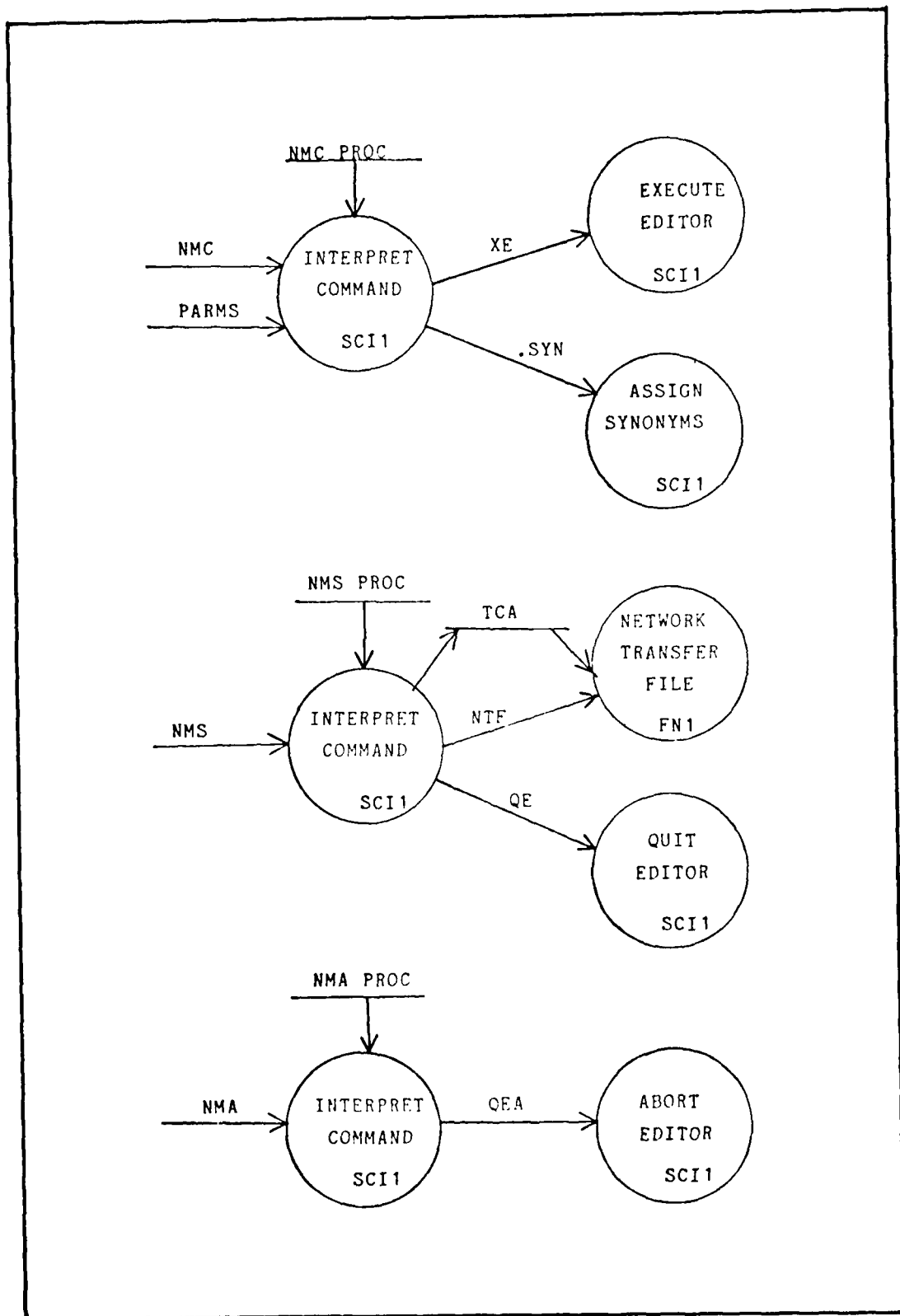


Figure 19. Network Message Commands

for future use with the other message commands. After composing the message, the operator strikes the terminal CMD key which allows him to enter one of two other network message commands.

Network Message Send (NMS). The NMS command creates the message file from the editor file and transfers the Message file to the message destination. As with the NMC command, the NMS command can be accomplished without unique R-S processes. However, one of the commands needed in the NMS PROC is the network command NTF. The PROC must contain the Quit Editor (QE) command which creates the message file from the Editor file and the NTF command which transfers the Message file to the message destination. The parameters needed with NTF are supplied by the synonyms set by the NMC command.

Network Message Abort (NMA). The NMA command discards the editor file if the operator decides not to send the message. Again all of the NMA functions can be accomplished with SCI commands and primitives in a PROC. The Quit Editor and Abort (QEA) command discards the Editor file. A status message can be sent to the Network Log file with a .DATA primitive.

Network Transfer Program File (NTPF)

The NTPF command results in program files being transferred from one site to another. In the DX-10

operating system, a program file is a relative record file that contains several executable, object code programs. The file contains an internal directory that enables direct access to individual file records. A program file is also categorized as a file directory in DX-10.

Before a program file can be transferred via the 3780 Emulators, the program file must be transformed into a sequential file. The DX-10 operating system accomplishes this transformation when commanded with a Backup Directory (BD) command. Conversely, when the sequential file arrives at the destination site, the sequential file must be transformed into a program file with a Restore Directory (RD) command.

Since the R-S processes at both ends of the link must be able to command the SCI, both background processes, BACKNET1 and BACKNET2, will have to participate in the execution of NTPF. The processes needed to realize the NTPF command are shown in figure 20.

To establish a liaison between BN1 and BN2 requires a more complicated initialization procedure than for the typical command. The FN1 Interpret Command process executes the BN2 process, and the BN2 process, in turn, executes the BN1 process. Once both BN1 and BN2 are activated, the initialization procedure between them is the same as between FN1 and BN2 in the typical command.

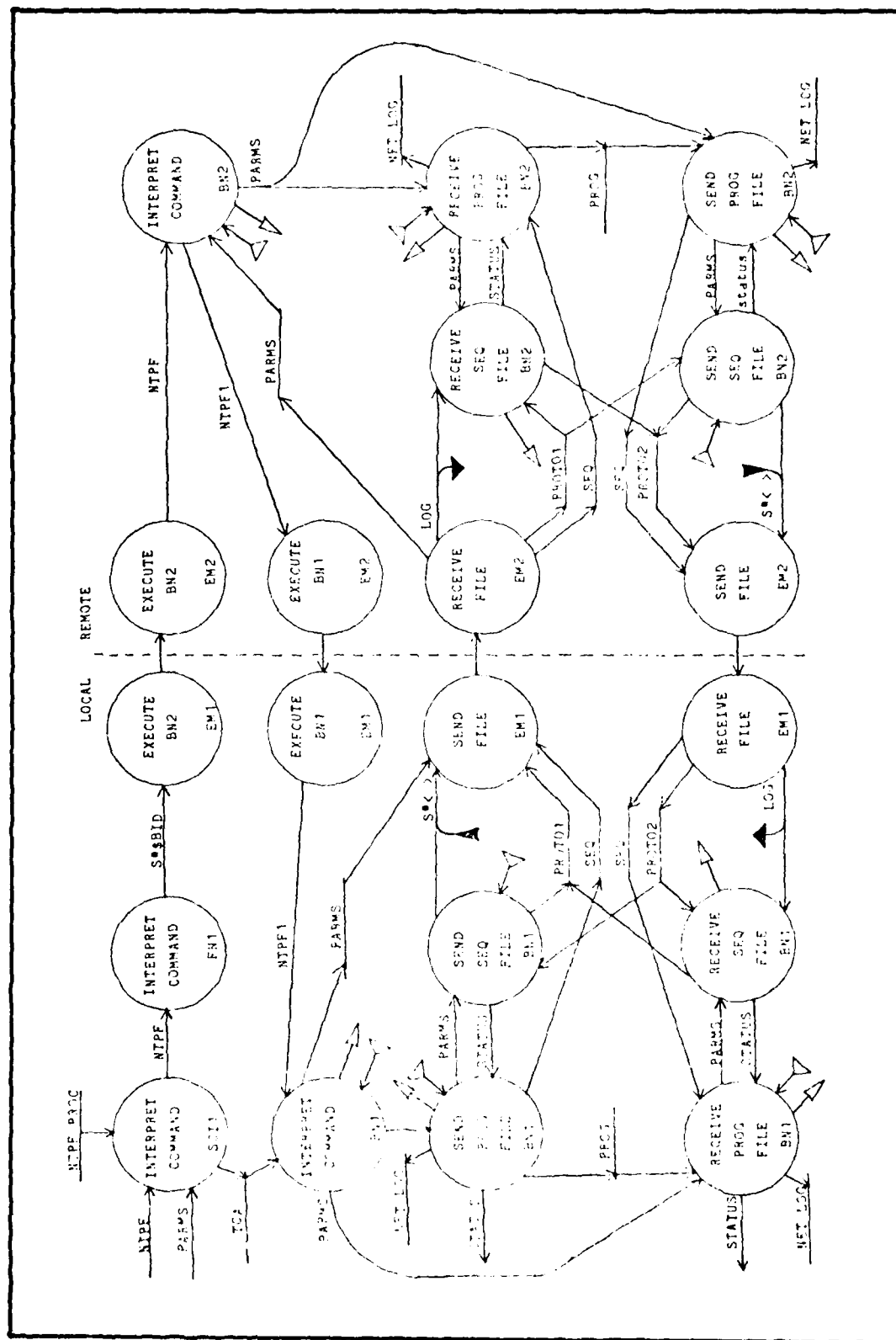


Figure 20. Network Transfer Program File (NTPF)

The Interpret Command processes of BN1 and BN2 must decide, based on the command parameters they both possess, which program file processes must be activated. If the program file is being transferred from the local end to the remote end of the link, the BN1 Send Program File and BN2 Receive Program File processes are activated. If the transfer is from the remote to the local end the processes are reversed. Once activated, the Program File processes establish communications and proceed to execute the necessary NTPF functions.

The Send Program File process must convert the program file to a sequential file before can be transferred to the other site. To effect the conversion, the Send Program File process executes a BD SCI command by constructing a BD Batch file and executing SCI. The BD Batch file contains a BD command and its associated parameters. The Send Program File process is expanded in figure 21 to illustrate the interface to SCI through S\$BIDT. When SCI finishes its execution of the BD Batch file, Send Program File checks the value of the \$\$CC system synonym to see if SCI has returned an error. If there is an error, the user is notified with a status message and the BN processes terminate. If there is no error, the Program File processes at each end of the link pass parameters to the Sequential File processes. The Send and Receive Sequential File processes cooperate as described

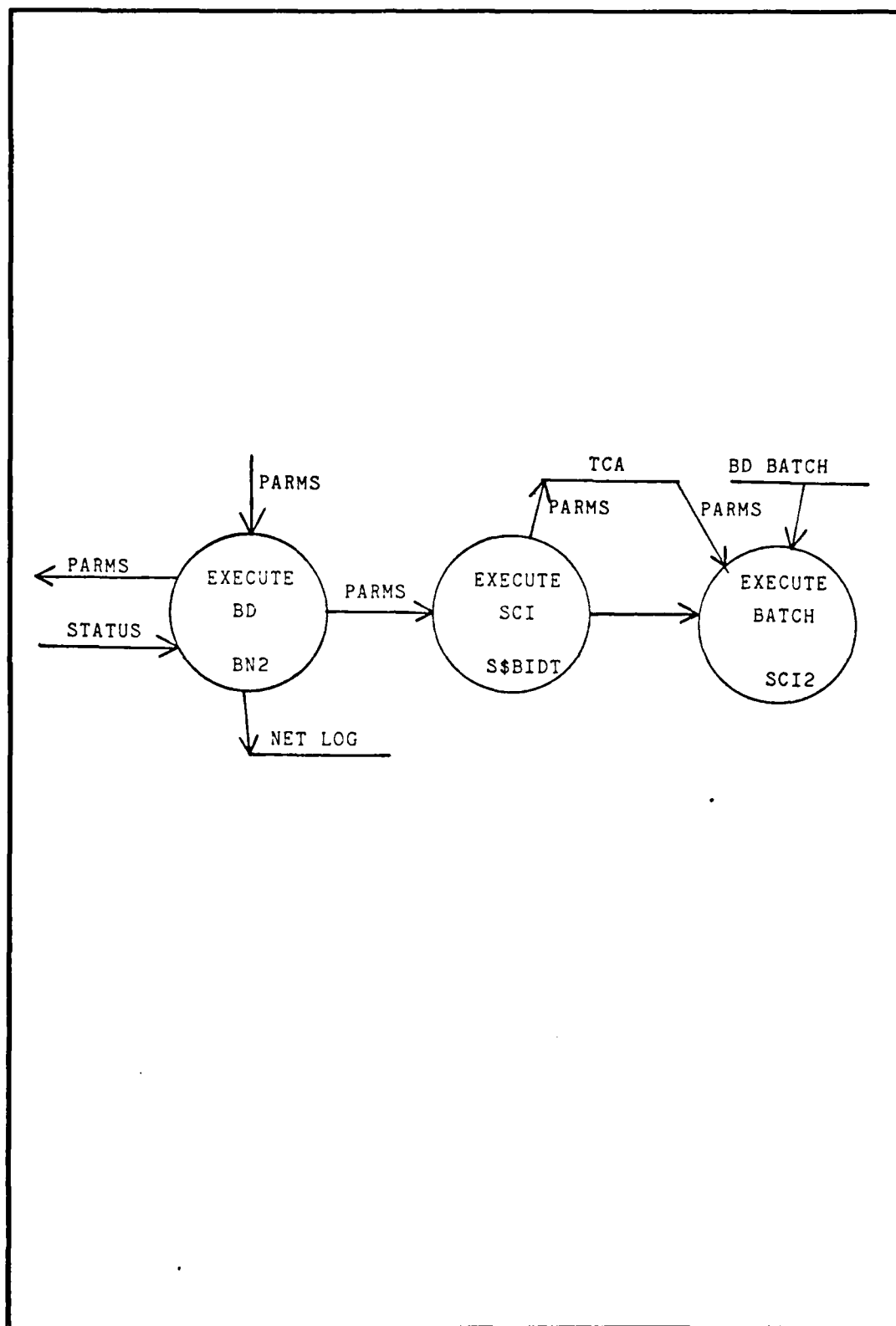


Figure 21. Send Program File

for the NTF command to transfer the sequential file across the link.

When the sequential file arrives at the opposite end of the link, it must be converted to a program file. The receive Program File process accomplishes the conversion by constructing a RD Batch file and executing SCI. An expanded view of the receive Program file process is shown in Figure 22.

Finally, the Program File processes make Network Log entries, send a status message to the user, and terminate.

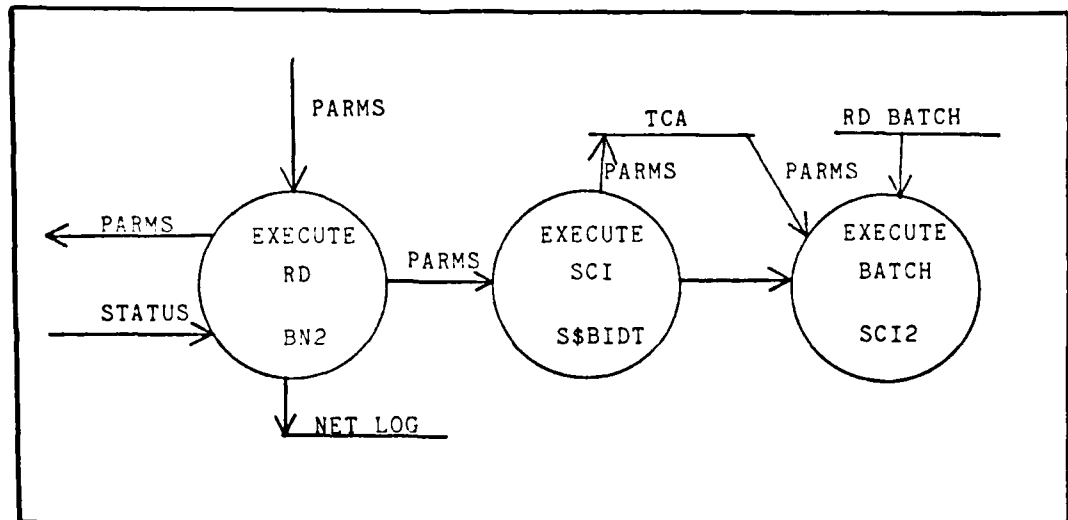


Figure 22. Receive Program File

Network Execute Task (NXT)

The Network Execute Task (NXT) command allows a user to execute any remote program on a remote site. NXT provides a much more general capability than does the Emulator. The 3780 Emulator can execute a remote task only if it is installed on the S\$PROGA system program file. The S\$PROGA program file is intended primarily for O/S programs, but it is sometimes justified to install an applications program in S\$PROGA. FORENET and BACKNET must be installed on S\$PROGA so that they can be executed with the Emulator bid task capability.

Figure 23 shows the NXT command processes. The initialization procedures end with the establishment of communications between the Execute Task processes of FN1 and BN2. The BN2 Execute Task process executes the specified task by constructing a NXT Batch file and executing SCI. BN2 Execute Task also checks the status of the executed task by periodically executing a Check Status of Tasks (STS) command through the S\$BIDT interface. When the executed task has terminated, BN2 Execute Task sends a completion message to FN1 and both R-S processes terminate.

Network Compile (NCOMP)

The NCOMP command give the user the capability to compile a source program at a remote site. If the source program originated at the local end of the link, it must first be transferred to the remote end with a NTF command. Similarly, if the object file resulting from the execution of NCOMP is to be executed at the local end, it must be transferred with the NTF command. The processes associated with NCOMP are illustrated in figure 24.

After the liaison between FN1 Compile and BN2 Compile is established, the Compile process constructs a NCOMP Batch file with the parameters which were acquired at the local end. Compile then executes SCI, specifying the NCOMP Batch file as the command source. Compile determines the status of the compile by reading the Compile Message file and reports status periodically to the user. When the compile is complete, the R-S processes terminate in their normal manner. The various files indicated in the DFD are available for subsequent network operations.

Network Link Edit (NLINK)

The NLINK command provides a remote link editing capability. The object files to be linked and the Link Control file must be resident at the remote site at the time the NLINK command is executed. The Link Control file contains commands that explicitly identify object files to be linked. The Link Control file also contains commands that identify libraries to be searched for object files referenced but not explicitly identified. The processes associated with NLINK are shown in figure 25.

The NLINK initialization procedure establishes a liaison between the FN1 and BN2 Link Edit processes. The BN2 Link Edit process constructs a NLINK Batch file that contains an Execute Link Edit (XLE) command and designates the relevant input and output files. BN2 Link Edit monitors the progress of the link edit by reading the Link Listing file. When the link edit is complete, BN2 Link Edit sends a status message to the user and the R-S processes terminate.

Network Execute Batch (NXB)

The NXB command results in a user specified batch file being executed at the remote site. NXB provides a general access capability to the resources of the remote site. NXB is the least automated of the network commands. To use the NXB command effectively, the operator must be proficient in the SCI language. The operator composes a batch file at the local site, transfers it to the remote site with the NTF command, and executes it with the NXB command. The processes needed for NXB are shown in figure 26.

After initialization of the FN1 and BN2, BN2 Execute Batch executes SCI2 through the S\$BIDT interface, designating the special batch file as the command source. BN2 Execute Batch checks the status of the batch execution by periodically executing a STS command and reading the STS file. When the batch execution is complete, the user is informed and FN1 and BN2 terminate.

Network Log Off (NLOFF)

The NLOFF command terminates the communications session between two sites by terminating communications between the Emulators and commanding the Emulators to the answer mode. The NLOFF command uses the parameters acquired by the NLON command. The processes associated with the NLOFF command are shown in figure 27.

After a liaison is established between the Log Off processes, both processes make Network Log entries with the user information acquired by the NLON command. Next, FN1 Log Off commands its Emulator to terminate with an emulator TERM command. Finally, both Log Off processes reactivate their Emulators, command the Emulators to the answer mode, and terminate themselves.

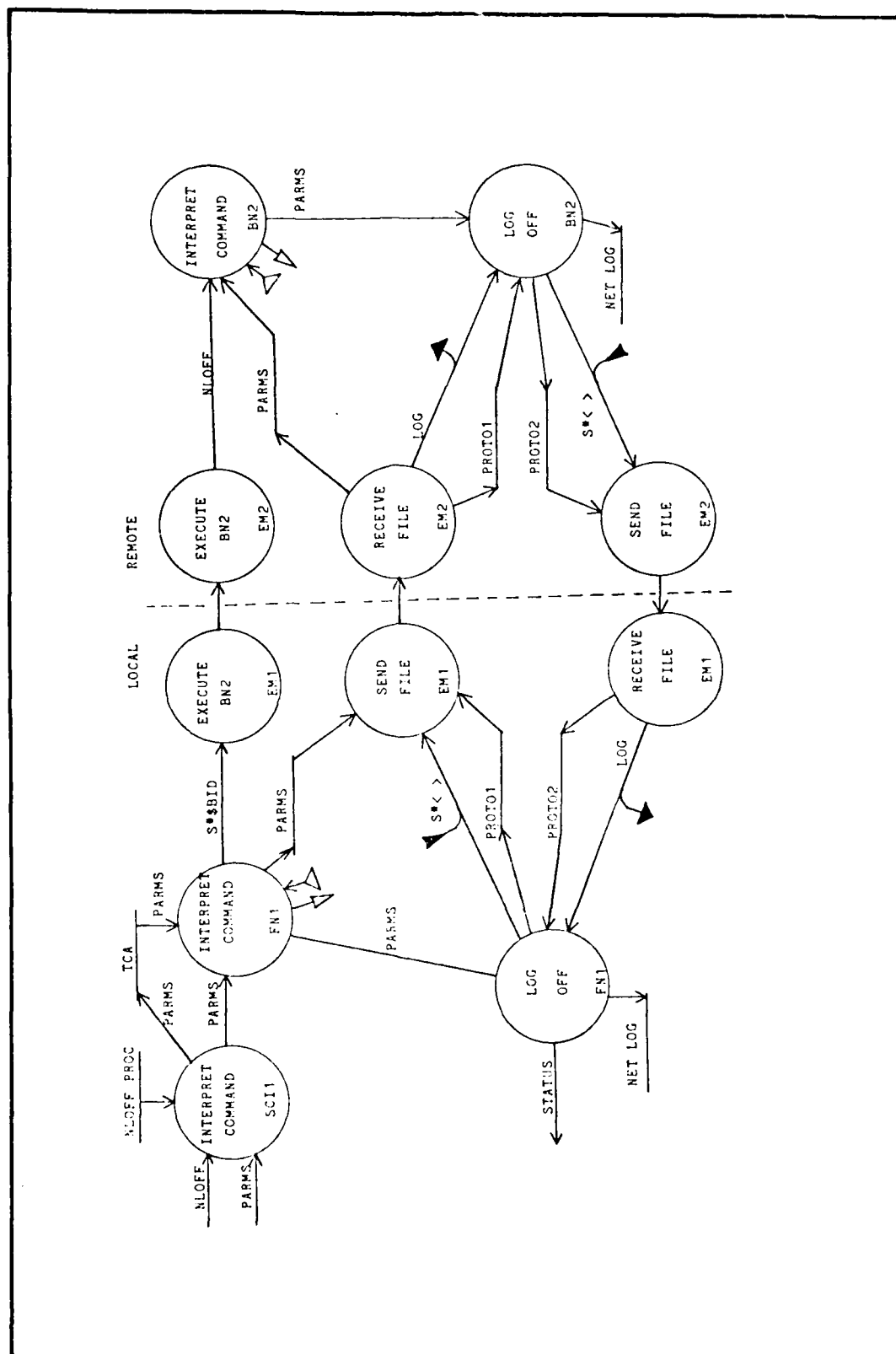


Figure 27. Network Log Off (NLOFF)

Network Quit (NQ)

The NQ command terminates the site's Emulator, ending the site's participation in the network. The processes needed for NQ are shown in figure 28. The Net Quit process logs the transaction in the Net Log, and terminates the Emulator with a TERM command.

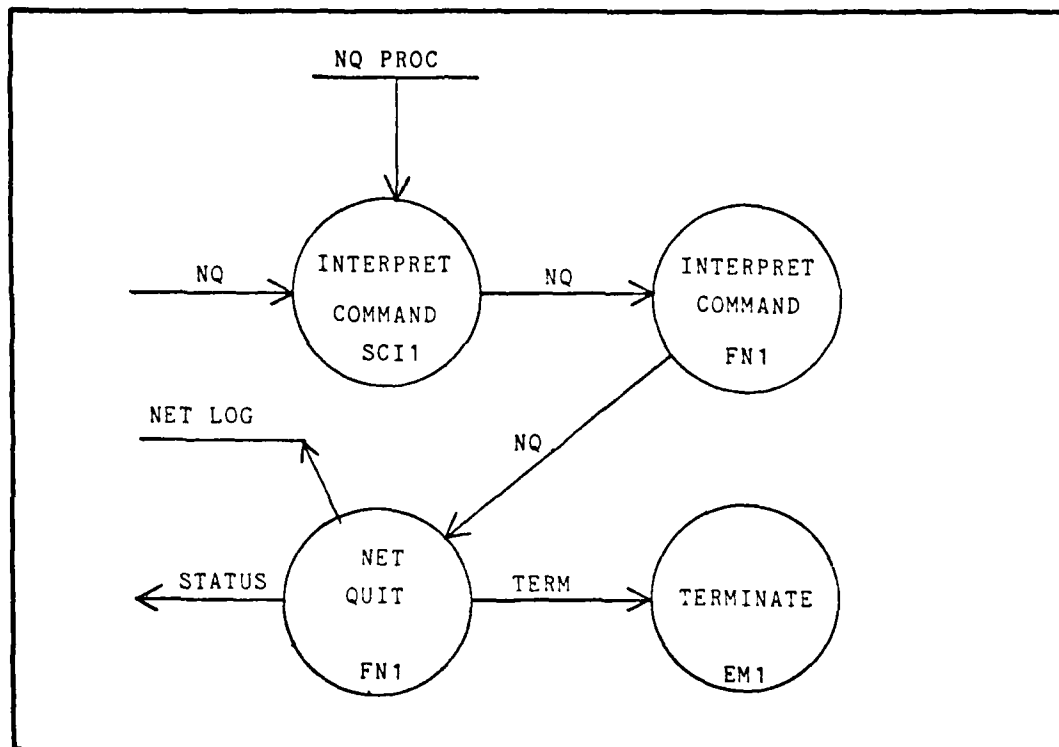


Figure 28. Network Quit (NQ)

Summary

This chapter has described the analysis of the thirteen R-S network commands. The analysis determined the processes that must be implemented with R-S software. The processes needed to realize each command were shown in separate data flow diagrams for each command. The processes were grouped in the data flow diagrams as they would be geographically, enhancing the visualization of the protocols described for each command. The next chapter describes the software design procedure of allocating to software modules the software functions defined in the analysis.

IV. Network Software Design

This chapter describes the design of the network software. The purpose of the design process was to arrive at an architecture for the software (Ref 11). The functions identified in the software requirements analysis process were allocated to software modules during the design process. The design assumed that the software would be implemented in the Pascal language, as dictated by the system requirements.

Structured Design

The structured design methodology was used to design the network software. The structured design process transforms data flow diagrams into a hierarchy of modules represented by structure charts (Ref 11). Structured design was chosen because the resulting structure charts communicate the design effectively.

Figure 29 illustrates an example of a simple structure chart. A box represents a software module that either transforms or processes data. A box with two vertical, interior lines represents an existing software module. The arrows connecting the modules represent calls to a subordinate module and the arrows beside the connecting arrows represent the transfer of data, control, or both.

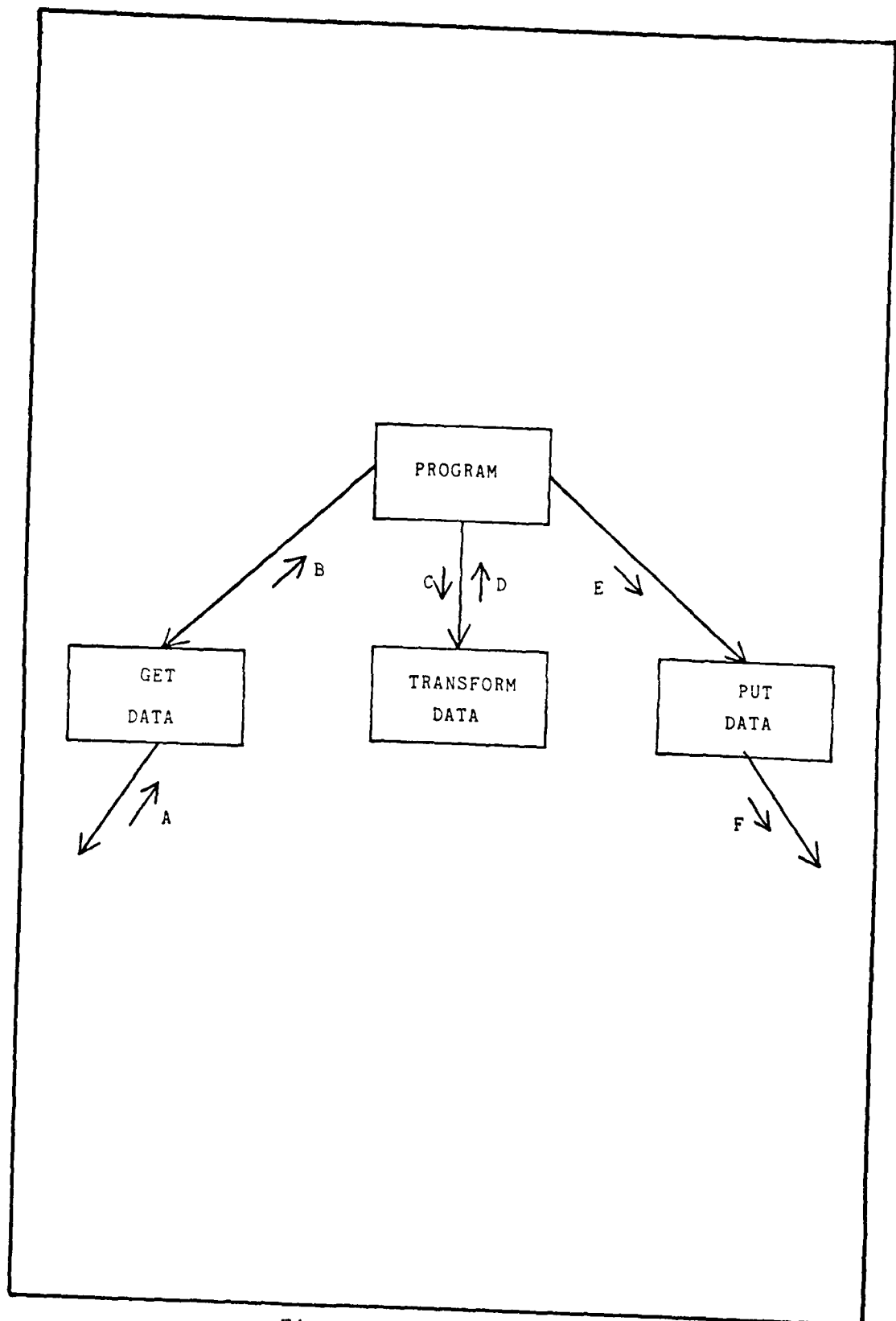


Figure 29. Structure Chart

The open circle represents existing operating system functions.

The basic structure illustrated in the figure is known as a source/transform/sink structure. Data is acquired through the left, or afferent, branch; transformed in middle, or transform, branch; and outputted through the right, or efferent, branch. The goal of structured design is to achieve a design that is readily understood. A design structure that reflects the problem it solves enhances understanding of the design (Ref 11:68). Therefore, if the basic nature of a problem is to input, process, and output data, then the source/transform/sink structure enhances understanding of the design by segregating the input/output functions from the primary functions of the program. Furthermore, the modules should be defined such that they implement one function that can be described by a verb-noun phrase, again enhancing understanding of the design.

Overview

Two software programs were designed as part of this investigation. The programs represent the static, instruction portion of the FORENET and BACKNET processes shown in figure 12, Chapter 2. Both programs reside in each AMS site, providing each site with the capability to participate in the execution of a network command. The structure charts that illustrate the design structure of

FORENET and BACKNET are shown in figures 30 and 31, respectively.

When one of the network programs is executed, its first actions involve the acquisition of a command and its parameters. Based on these inputs, the Initialize Link module can initialize the network as required for the particular network command. The Initialize Link module is an example of a transform module as defined in the previous section. By processing the input data, Initialize Link determines whether the Execute BN1 and Send Parms modules should be called to establish a link with another site. If a distant program is executed, Initialize Link must also decide which data is to be passed to the program. Once the link is initialized for a particular network command, one of the remaining 10 transform modules are selected to perform the desired resource-sharing network functions.

After a resource-sharing module is called and parameters have been passed to the module, the module retains control until all network command functions have been accomplished. No additional data transactions are handled by the apparent afferent and efferent branches. This structure may at first sight appear to violate the goals of structured design and, in particular, the source/transform/sink structure. However, the design structure does support the stated goal of reflecting the

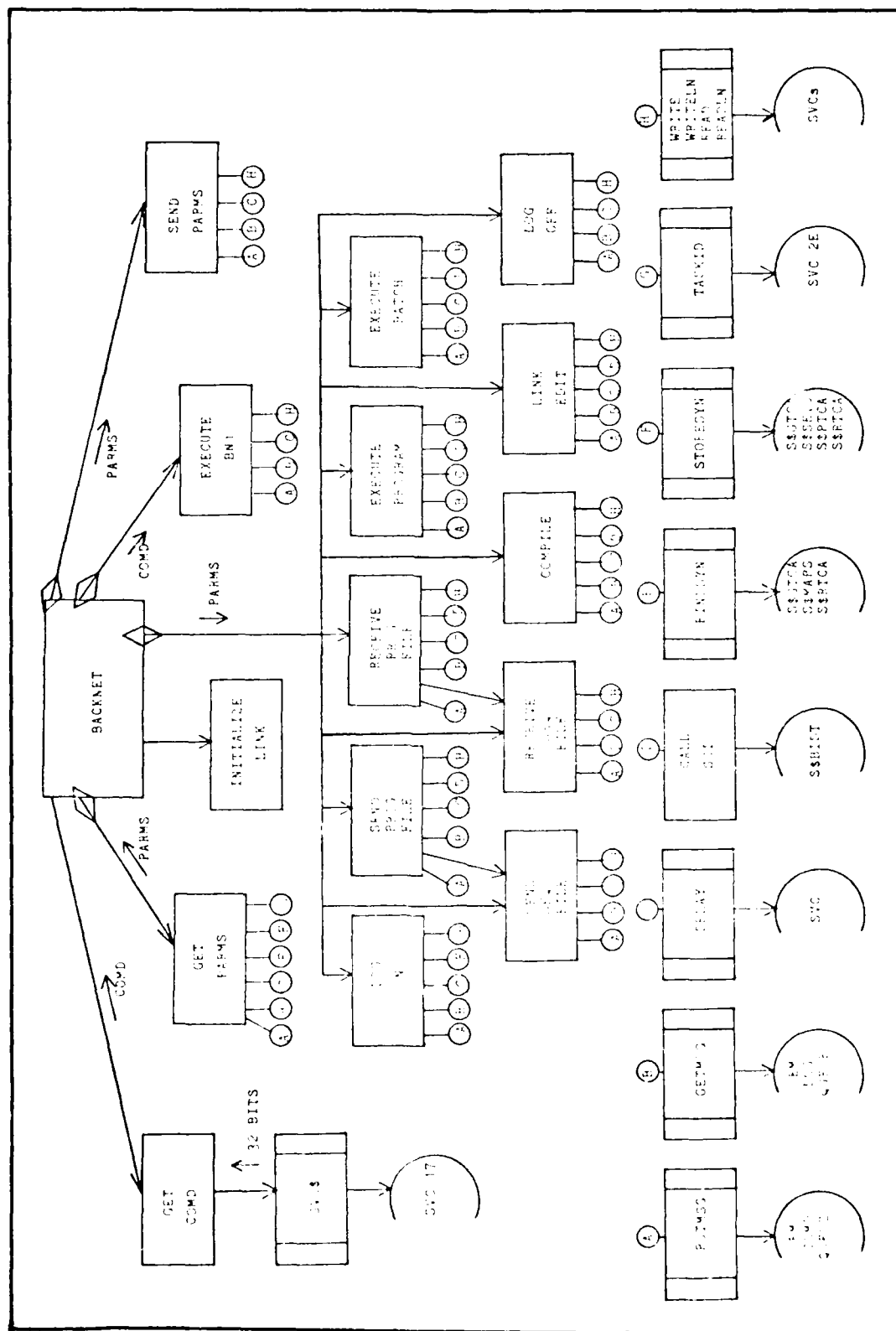


Figure 31. BACKNET Structure Chart

nature of the problem. A resource-sharing (R-S) module is called to accomplish a particular R-S function and most R-S functions inherently require communications (input/output). Therefore, the R-S modules are provided with direct control over the communications resources needed to accomplish their functions. When a module has completed its assigned function, control returns to the main module and the program terminates.

O/S Interface Procedures

Most of the network software modules must interface with the DX-10 operating system. The interfaces are implemented with various Pascal procedures. Most of the procedures are DX-10 operating system dependent, external Pascal procedures. One of the O/S dependent external procedures, Call SCI, was defined by this investigation and the remainder are supplied with the DX-10 O/S. There are also several standard procedures that are automatically included at compile time to implement the many standard characteristics of the TI-Pascal language. A few of the standard procedures will be shown on the structure charts to emphasize the functions they provide.

The O/S interface procedures appear along the bottom of the structure charts. Connections between the interface procedures and their calling modules are indicated by the alphabetic code in the small circles. O/S modules called by

AD-A100 793

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCH00--ETC F/6 9/2
DESIGN OF A RESOURCE-SHARING NETWORK LINK.(U)

DEC 80 T M MCLEOD

AFIT/6E/EE/80D-30

NL

UNCLASSIFIED

2
4-
100-183



END

DATE

FILMED

7 81

DTIC

the interface procedures are indicated in the large open circles. The functions of the interface modules are described below.

Putmsg. Putmsg is a DX-10 dependent external Pascal procedure that provides an interface with the Emulator Command Queue. A packed array of variable size is passed to the procedure. The procedure places the array in the Emulator command Intertask Message Queue (IMQ). Putmsg uses Supervisor Call (SVC) 1C.

Getmsg. Getmsg is a DX-10 dependent external Pascal procedure that provides an interface with the Emulator log IMQ. The procedure fetches a message from the IMQ and passes it to the calling module. Getmsg uses SVC 1D.

Delay. Delay is a DX-10 dependent external Pascal procedure that suspends the calling program for a period equal to some multiple of 50 milliseconds (ms). The procedure call is typically used when the calling module wants to wait for a period time before checking for the arrival of a protocol message. Use of the Delay procedure reduces the amount of processor time expended while waiting for a response. Otherwise, the R-S command module would continually attempt to check the Log Queue for the arrival of the Protocol file. The Delay procedure uses SVC 02.

Read and Write. The Read, Readln, Write, and Writeln procedures are standard TI-Pascal procedures needed to

implement the basic TI-Pascal language. They are included in the structure charts to emphasize that the R-S command modules must access data files and devices.

Call SCI. Call SCI is an external Pascal procedure that was defined by this investigation. The procedure interfaces with S\$BIDT to provide a means of executing SCI in the batch mode as described in previous chapters.

Findsyn. Findsyn is a DX-10 dependent, external Pascal procedure that returns the value of a system or user defined synonym. Synonyms provide a convenient means of storing parameters that will be constant from one network command to another. Information that is constant from one network command to another is stored in synonyms. Another advantage of synonyms is that they can be used as variables in Command Procedure (PROC) files.

Findsyn calls O/S modules S\$GTCA, S\$SETS, S\$PTCA, and S\$RTCA. S\$GTCA (Get Terminal Communications Area) opens the TCA to other S\$ modules. S\$MAPS finds the requested synonym and returns its value. S\$RTCA (Release TCA) closes the TCA.

Storesyn. Storesyn is a DX-10 dependent, external Pascal procedure that sets a synonym to a requested value. Storesyn is used to set synonym values that are subsequently fetched using Findsyn. Storesyn uses O/S modules S\$GTCA, S\$SETS, S\$PTCA, and S\$RTCA. S\$SETS (Set Synonym) sets a

synonym to a newly specified value. S\$PTCA (Put TCA) installs the new or changed synonym in the TCA.

Taskid. Taskid is a DX-10 dependent, external Pascal procedure that fetches the runtime ID of the calling task. FORENET and BACKNET use their runtime IDs as a unique identifier of the Log Queue. Taskid uses SVC 2E.

The preceding paragraphs under "Overview" have described characteristics of the network software design that are common to FORENET and BACKNET. The next two sections describe characteristics that are unique to FORENET or BACKNET.

FORENET

FORENET is the local program responsible for cooperating in the execution of all network commands except Network Transfer Program File (NTPF). The local BACKNET (BN1) participates in the execution of NTPF. Upon execution, FORENET fetches commands and parameters. For all commands except Network Initialize (NI) and Network Quit (NQ), BN2 is executed with S*\$BID Emulator command and parameters are passed to BN2 with the Parms file. NI and NQ are performed locally only, and, therefore, they do not need the cooperation of BN2. If the command is NTPF, FORENET terminates after executing BN2. For all commands except NTPF and after the link has been initialized, control is passed to one of the R-S command modules which performs the

required command functions.

All of the FORENET modules except the O/S interface modules are described briefly in this section. There is a one-to-one correspondence between most of the modules and the processes defined by the software analysis in Chapter 3. The only exception is that the Interpret Command process has been divided into four modules: Get Comd, Get Parms, Execute BN2, and Send Parms.

Get Comd. The first action of FORENET is to call Get Comd. The Get Comd fetches an integer that represents the network command from a special TCA storage location. The storing of the integer is an ancilliary result of the The .BID primitive is followed by a SCI variable known as "CODE". CODE is set to an integer value representing the network command. For instance, CODE is set to 1 for NI, 2 for NLON, and 3 for NTF. Get Comd fetches the value of CODE by calling S\$GTCA to gain access to the TCA, S\$STAT to get the value of CODE, and S\$RTCA to release the TCA.

Get Parms. The Get Parms module fetches the parameters stored in the TCA by SCI. Again, access is gained to TCA with a call to S\$GTCA. Subsequently, S\$PARMS is called to get the value of a parameter. The parameters are stored in a sequential order corresponding to the order in which they were acquired. A particular parameter is fetched by passing an integer that corresponds with the command parameter's

placement in the TCA parameter table.

Get Parms also fetches the runtime IDs of FORENET and the Emulator. Get Parms calls Taskid and Findsyn to get the runtime IDs of FORENET and Emulator, respectively. Get Parms will not call Findsyn if the command is NI because the synonym for the Emulator runtime ID will not have been set by the Log On module yet.

Initialize Link. Initialize Link sets a control variable that indicates whether Execute BN2 or Send Parms should be called. Execute BN2 is called for all commands except NI and NQ. Send Parms is called for all command except NI, NQ, and NTPF.

Execute BN2. BN2 is executed with a S*\$BID command to the Emulator. The S*\$BID command includes a 32 bit word that becomes available to BN2 as a special parameter. Execute BN2 sets the parameter to the same integer value as CODE.

Initialize Emulator. This module initializes the Emulator by commanding it with an ANS (answer) command. The Emulator is executed prior to the call to Initialize Emulator by a X3780 SCI command in the NI PROC. The NI PROC also contains a STS command that results in the Emulator status being placed in the STS List file. Initialize Emulator determines the runtime ID of the Emulator from the STS List file and sets a synonym to the runtime ID value by calling Storesyn. The module communicates status to the user and the Net Log,

and terminates.

Log On. The Log On module records the user information in the Net Log and communicates with the remote Log On to verify that the log on activities are being accomplished at the remote end as well. When the command functions are complete, status is communicated and the module terminates.

Send Sequential File. The Send Sequential File module sends a file to the remote site by commanding the Emulator to send the specified file. After the module receives confirmation from the remote, cooperating module that the file has arrived properly, status is logged and the module terminates.

Receive Sequential File. The FORENET Receive Sequential File is called when FORENET has requested that a remote file be transferred to the local site. The module communicates with remote module during the transfer and monitors the Log Queue to determine when the transfer is complete. When the transfer is complete, Receive Sequential File notifies the remote module and follows the normal termination procedure.

Execute Task. The Execute Task module monitors the progress of the cooperating module at the remote site. The module reports status to the user periodically. When the remote module reports that the remotely executed user program has completed execution, the FORENET Execute Task module logs status and terminates.

Execute Batch. The Execute Batch module monitors the activities of the BN2 Execute Batch module. When the module receives a message from BN2 indicating that the User Batch file has completed execution, Execute Batch sends its usual status messages and terminates.

Compile. The Compile module monitors its complementary module at the remote site. Communications are maintained between the modules while the remote module accomplishes the primary objectives of the NCOMP command. When the remote compile is complete, the Compile module terminates.

Link Edit. The Link Edit module monitors the actions of the remote Link Edit module. When the link edit is complete, the module terminates.

Log Off. The Log Off module records user information and terminates communications between the Emulators. The user information that must be logged is obtained by calling Findsyn. Emulator communications are terminated by commanding the Emulator with a TERM command. The TERM command also terminates execution of Emulator, necessitating reinitialization of the Emulator with a NI SCI command. The NI command is contained in the NLOFF PROC after the .BID primitive and, therefore, is executed after the Log Off module has terminated.

Terminate Emulator. The Terminate Emulator module terminates the Emulator by commanding the Emulator with a

TERM command.

BACKNET

The BACKNET program usually executes as the remote cooperating program, but, when the NTPF command is received, both the local and remote BACKNETs must be executed. The local BACKNET is known as BN1 and the remote BACKNET is known as BN2. All of the BACKNET modules except the O/S interface modules are described in this section.

Get Comd. The Get Comd module fetches the 32 bit word that arrived in conjunction with the S*\$BID command. The word represents the network command and is fetched by calling the SVC\$ module. The SVC\$ module uses the Get Parameter Supervisor Call (SVC 17) to obtain the word.

Get Parms. The Get Parms receives the Parms file from either FN1 or BN1. The parameters are always passed from the local to the remote end of the link. For all commands except NTPF the file is passed from FN1 to BN2. For NTPF the file is passed from BN1 to BN2.

Initialize Link. The Initialize Link module processes the command and its parameters which were received from the local site to determine whether the Execute BN1 and Send Parms modules should be called. The modules are called only if the command is NTPF.

Log On. The BACKNET Log On module performs the same functions as the FORENET Log On module.

Send Program File. The Send Program File module resides in BACKNET only. The module passes a program file from the local to the remote end of the link if the module is in BN1, or vice versa, if the module is in BN2. In either case, the module executes a Backup Directory (BD) Batch file by calling Call SCI. At the completion of the BD Batch file, the specified program file has been converted to a sequential file, and parameters and control are passed to the Send Sequential File module. The Send Sequential File cooperates with its complementary module to transfer the sequential file across the link. When the sequential file has been received at the opposite end of the link, control returns to the Send Program File which monitors its complementary module executing a Restore Directory (RD) Batch file. When the program file has been properly installed, the module terminates.

Receive Program File. The Receive Program File module resides in BACKNET only and participates in the transfer of a program file. The module monitors the execution of the BD Batch file by its cooperating module. When the program file has been converted to a sequential file, parameters and control are passed to the Recieve Sequential File module. When the transfer of the sequential file is complete, control returns to the Receive Program File module which executes a RD Batch file. The completion of the RD Batch

execution marks the end of the NTPF command functions and the module terminates.

Send Sequential File. The BACKNET Send Sequential File module cooperates with a Receive Sequential File in FORENET or BACKNET, depending on whether it is executing a NTF or NTPF command.

Receive Sequential File. The BACKNET Receive Sequential File module cooperates with a Send Sequential File module in FORENET or BACKNET, depending on whether it is executing a NTF or NTPF.

Execute Task. The primary objectives of the NXT command are accomplished in the BACKNET Execute Task module. The module executes the NXT Batch file by calling the Call SCI module. The Module periodically checks the status of the task execution by executing a STS Batch file and reading the STS List file. The module keeps the cooperating module informed of the progress of the execution through frequent communications via the Protocol files. When the task execution has completed, the module sends a status message to the cooperating module and terminates.

Execute Batch. The Execute Batch module executes the designated User Batch file by calling the Call SCI module. The module monitors the batch execution, sends status to the cooperating module, and terminates when all command functions have been accomplished.

Compile. The Compile module builds a NCOMP Batch file with the NCOMP command parameters and executes the batch file by calling Call SCI. Compile monitors the progress of the compile by reading the Compile Message file. When the compile is complete, the module follows the normal termination procedure.

Link Edit. The Link Edit module builds a NLINK Batch file and executes it by calling the Call SCI module. Link Edit monitors progress by reading the Link List file. When the Link Edit is complete, the module terminates.

Log Off. The BACKNET Log Off module performs the same functions as the FORENET Log Off module.

Summary

This chapter has described the design of the network software programs FORENET AND BACKNET. The structure of the design was illustrated with structure charts that depicted program modules and their interrelations. Modules were defined to perform major functions such as executing a particular network command. Each module that implemented R-S functions was given control of its input and output functions. By controlling its communications, a module achieves the independence it needs to execute R-S functions efficiently.

V. Results and Recommendations

The primary objective of this investigation was to design a resource-sharing (R-S) network link that would allow an operator or user program to access remote computer resources by entering simple network commands at the local computer. The design was constrained to use existing hardware and software modules as applicable. An R-S network link design that was specialized to accommodate existing model 990 computers was accomplished by this investigation. The design results are summarized in the section. The final section contains recommendations for refining and implementing the design.

Design Results

The design that resulted from this investigation consists predominantly of existing hardware and software modules. The goal of the design was to combine the existing capabilities in a manner that made them appear to the user to be two comprehensive, cooperating R-S processes capable of performing all of the required functions. Two new modules, the programs FORENET and BACKNET, were designed to accomplish the task of coordinating, or automating, the execution of the existing capabilities. Also, a new interface, the S\$BIDT interface, was designed that allows access to any resource controlled by the DX-10 operating

system.

The R-S programs FORENET and BACKNET perform the required R-S functions by coordinating the execution of capabilities under their control. The programs are able to coordinate their activities by exchanging messages as prescribed by R-S protocols.

Communications are implemented through a combination of several existing modules. The most important of these modules is the 3780 Emulator. The Emulator provides the communications interface to the R-S programs. The basic capability of the Emulator is to transfer files. Therefore, the communications technique used by the R-S programs is to place protocol messages in various files and to command the Emulator to transfer the files. The protocol messages enable each program to decide which protocol state to enter next. A program determines its next action by comparing its current state and the messages received from its cooperating process against the protocols, or logical rules, that are inherent in the logical structure of the program.

The R-S network link designed is capable of performing complex R-S functions after receiving only a simple mnemonic command. The cooperating R-S processes actively communicate to coordinate the execution of numerous subfunctions necessary to fulfill the goals of the requested R-S function. If an error is detected the processes attempt to

recover. However, if recovery attempts also fail the processes proceed to achieve a well defined abnormal termination and report the error to the user.

Recommendations

The primary recommendation is that the design contained in this report be implemented. Throughout the design, ease of implementation was a major consideration.

The network commands were defined in a manner very similar to the the operating system (SCI) commands are implemented. The advantage of defining the network commands in that manner is that ample documentation (Ref 18) is available to aid in the implementation. Similarly, the R-S programs were refined as Command Processors as defined in operating system documentation. Numerous operating system routines that enhance interactive operation are available to programs that are so defined.

Before FORENET and BACKNET are implemented it is recommended that the modularization defined in Chapter 4 be extended to at least one more level. The functions that must be carried out by each module are complex enough that a single corresponding Pascal procedure would be difficult to read and, therefore, would also be difficult to test and debug.

It is also recommended that the protocol designs described in Chapter 3 be refined. The protocols described

represent a preliminary design. Since protocols determine the flow of control between cooperating processes, they should be more precisely documented before an attempt is made to code them. References 6 and 16 provide excellent background information on the subject of data communication protocol design.

BIBLIOGRAPHY

1. Aeronautical Systems Division/AC. Management Approach for the ASD Automated Management Systems Program (Draft Memorandum). Wright-Patterson AFB, Ohio: ASD/AC, 10 May 1979.
2. Abrams, Marshall, Robert P. Blanc, and Ira W. Cotton. Computer Networks: A Tutorial. New York: Institute of Electrical and Electronics Engineers, Inc., 1978.
3. Allen, Brian C., Systems Analyst. Personal interviews. Aeronautical Systems Division/ADDI, Wright-Patterson AFB, Ohio, 1980.
4. Chu, Wesley W. Advances in Computer Communications. Dedham, Massachusetts: Artech House, 1977.
5. Cooper, Ted, Systems Analyst. Personal interviews. Microbase Company, Dayton, Ohio. 1980.
6. Davies, Donald W. et al. Computer Networks and Their Protocols. New York: John Wiley and Sons, 1979.
7. Davies, Donald W. and Derek L. A. Barber. Communication Networks for computers. New York: John Wiley and Sons, 1973.
8. Feinler, Elizabeth and Jonathon Postel. Arpanet Protocol Handbook. San Francisco, California: Data Composition, An Arcata National Company, 1978. (AD A052 594)
9. Grogono, Peter. Programming in Pascal. Menlo Park, California: Addison-Wesley Publishing Co. Inc., 1978.
10. Itzkowitz, Avrum E. A Survey of the Properties of Computer communication Protocols, Vol I: The Functions, Properties, Specification, and Analysis Methods of Computer Communication Protocols. Champaign, Illinois: U. S. Army Construction Engineering Research Laboratory, September 1978. (AD A063 093)
11. Jensen, Randall W. and Charles C. Tonies. Software Engineering. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979.

12. Kahn, Robert E. "Resource-Sharing Computer Communications Networks, "Proc. IEEE, 60 (11): 1397-1407 (November 1972).
13. Karp, Harry R. Basics of Data Communications. New York: McGraw-Hill Publications Co., 1976.
14. McQuillan, John M. and Vinton G. Cerf. Tutorial: A Practical View of Computer Communications Protocols. Long Beach, California: IEEE Computer Society, 1978.
15. Myers, Glenford J. Composite/Structured Design. New York: Van Nostrand Reinhold Co., 1978.
16. Postel, Jonathon B. A Graph Model Analysis of Computer Communication Protocols. PhD. dissertation. Los Angeles, California: University of California, Los Angeles, January 1974. (AD 777 506)
17. Sunshine, Carl. A. Interprocess Communication Protocols for Computer Networks. PhD. dissertation. Stanford, California: Stanford University, December 1975. (AD A025 508)
18. Texas Instruments Inc., Digital Systems Division. Model 990 Computer DX10 Operating System Release 3, Volumes 1 through 6. Austin, Texas: Texas Instruments Inc., 1978.
19. Texas Instruments Inc., Digital Systems Division. TI Pascal Users Manual. Austin, Texas: Texas Instruments Inc., 1979.
20. Texas Instruments Inc., Digital Systems Division. Model 990 DX 3780/2780 Emulator user's Guide. Austin, Texas: Texas Instruments Inc., 1980.
21. Wecker, Stuart. "Computer Network Architectures," Computer, 12 (9): 58-72 (September 1979).
22. Yourdon, Edward and Larry L. Constantine. Structured Design. New York: Yourdon Inc., 1975.
23. Zimmerman, Hubert. "OSI Reference Model--The ISO Model of Architecture for Open Systems Interconnection," IEEE Transactions on Communications, COMM-28 (4): 425-432 (April 1980).

Appendix

This appendix contains the program listings used to demonstrate the S\$BIDT/SCI interface.

The first program listing on pages 111 and 112 is the Pascal program CALLP\$ that demonstrates how a Pascal program calls the procedure Call SCI, defined previously in Chapter 4. The external procedure SYSBID found in the Pascal source listing corresponds to Call SCI.

The next program on pages 113 through 115 is an assembly code listing which is structured as the TI Pascal Compiler would structure an external procedure (Ref 19). When CALLP\$ calls SYSBID, the parameters are placed in locations determined by compiler characteristics. SYSBID "knows" where the parameters are placed and rearranges them to locations where S\$BIDT expects to find them.

The listing on page 116 is the demonstration data read by CALLP\$.

The listing on page 117 is the contents of the batch file that SCI reads.

The listing on page 118 is the batch listing which results from SCI reading the batch file. The batch file contains a .SHOW primitive designating the data file which was read by the Pascal program. The .SHOW primitive causes the indicated file to be read into the batch list. The data

on page 116 can be found in the batch list, demonstrating that SCI was executed in the batch mode.

The following information is taken directly from a TI information sheet:

S\$BIDT SUBROUTINE

Cooperating tasks written in assembly language that do not call the S\$ subroutines may execute each other by including an Execute Task SVC in the calling task. When both tasks are in the same program file, the LUNO is already assigned. Otherwise, the LUNO may be assigned by an interactive SCI command, by a batch mode SCI command, or by an I/O Utility SVC in the calling task (preceding the Execute Task SVC).

The S\$BIDT subrouting should be used to call a cooperating task that calls S\$ subroutines. Either S\$GTCA or S\$NEW must be executed prior to calling S\$BIDT. The S\$BIDT subroutine uses R0 to return a completion code. A value of zero is returned when the subroutine completes satisfactorily; a nonzero value is returned when an error has occurred. Prior to calling S\$BIDT, the user must place the following values in registers R1 through R3:

R1, left byte - Task ID of task to executed.

R1, right byte - LUNO of program file that contains the task.

R2 - Address of parameter table, or zero when there are

no parameters.

R3, left byte - Code value, or zero.

R3, right byte - Flags:

Bits 8-11 - Set to zero.

Bits 12 - set to one.

Bits 13-15 - set to zero. The parameter table address in R2 is the address of a table of words that contain addresses, followed by a word that contains zero. Each address in the parameter table is the address of a parameter in the following format:

Byte 0 - Number of characters in parameter.

Byte 1 through n - characters of parameter. When a parameter has a null value, set byte 0 of the parameter to zero.

The code value in the left byte of R3 is an integer in the range of 0 through 255 that may be accessed as a binary value by the called task. If the task does not require a code value, enter zero.

Subroutine S\$BIDT executes an Execute Task Supervisor call. The values in R1 and the right byte of R3 are placed in the appropriate bytes of the supervisor call block. That is, the task ID, LUNO, and flags are those values required for an Execute Task SVC. The SVC also passes the parameters and

the code value to the called task. The called task obtains the parameters by calling the P\$PARM subroutine.

Subroutine S\$BIDT is called as follows:

BLWP @S\$BIDT

```
(*$WIDELIST*)  
PROGRAM CALLP3;
```

```
CONST  
  SL=40;  
  LENGTH=39;
```

```
TYPE  
  STRING=PACKED ARRAY[1..SL] OF CHAR;
```

```
VAR  
  CMDFILE: STRING;  
  LSTFILE: STRING;  
  FLAG: INTEGER;  
  TASKID: INTEGER;  
  CODE: INTEGER;  
  ERROR: INTEGER;  
  NUMB: INTEGER;  
  PFLUNO: INTEGER;  
  SRCFILE: TEXT;  
  V: INTEGER;  
  S: INTEGER;
```

```
PROCEDURE SYSBID (TASKID: INTEGER; PFLUNO: INTEGER;  
  INACNM: STRING; LACNM: STRING;  
  CODE: INTEGER; VAR ERROR: INTEGER;  
  LENGTH: INTEGER; FLAG: INTEGER); EXTERNAL;
```

```
PROCEDURE IOBID (TASKID: INTEGER; PFLUNO: INTEGER; VAR INACNM: STRING;  
  VAR LSTACNM: STRING; CODE: INTEGER; VAR ERROR: INTEGER;  
  LENGTH: INTEGER);
```

```
PROCEDURE SHIFTOE (VAR STR: STRING);
```

```
  CONST  
    BLANK = ' ';
```

```
  VAR  
    NEW: STRING;
```

```
  BEGIN
```

```
    FOR N := 1 TO SL DO  
      NEW[N] := BLANK;  
    S := 1;  
    V := 2;  
    FOR N := 1 TO SL-1 DO  
      BEGIN  
        NEW[V] := STR[S];  
        S := S+1;  
        V := V+1;  
      END;
```

```
    STR := NEW  
  END;
```

```
  BEGIN  
    SHIFTOE(INACNM);
```

```

WRITELN( INACNM=1, INACNM);
SHIFTONE(LETOCNM);
WRITELN(LETOCNM=1, LETOCNM);

```

END;

BEGIN

```

RESET(SRCFILE);
IF NOT EOF(SRCFILE) THEN
  READLN(SRCFILE, TASKID);
IF NOT EOF(SRCFILE) THEN
  READLN(SRCFILE, PFLUNO);
IF NOT EOF(SRCFILE) THEN
  READLN(SRCFILE, CMDFILE);
IF NOT EOF(SRCFILE) THEN
  READLN(SRCFILE, LSTFILE);
IF NOT EOF(SRCFILE) THEN
  READLN(SRCFILE, CODE);
IF NOT EOF(SRCFILE) THEN
  READLN(SRCFILE, FLAG);

```

```

WRITELN( TASKID=1, TASKID:3, PFLUNO=1, PFLUNO:3);
WRITE( CMDFILE=1);
WRITELN(CMDFILE);
WRITE( LSTFILE=1);
WRITELN(LSTFILE);
WRITELN( CODE=1, CODE:3, LENGTH=1, LENGTH:3, FLAG=1, FLAG:3);
IOBID (TASKID, PFLUNO, CMDFILE, LSTFILE, CODE, ERROR, LENGTH);
SYSBID(TASKID, PFLUNO, CMDFILE, LSTFILE, CODE, ERROR, LENGTH, FLAG);
WRITELN (ERROR=1, ERROR);

```

END.

IDT 'SYSBID'

OPEN DATA 0
BYTE 0,0A1
DATA 0,0,0,0

BINHEX BYTE 00
BYTE 0
OUTHEX BYTE 0,0,0,0

WRTHEX DATA 0
DATA 0BA1
DATA 0
DATA @OUTHEX
DATA 0
DATA 4

WRTCR DATA 0
DATA 0BA1
DATA 0
DATA @CRT
DATA 0
DATA 2

CRT DATA 00A0D

WRTIN DATA 0
DATA 0BA1
DATA 0
DATA 0
DATA 0
DATA 40

WRTL DATA 0
DATA 0BA1
DATA 0
DATA 0
DATA 0
DATA 40

BOT EQU 80
ARG EQU 100
TASKID EQU ARG+0
PFLUND EQU ARG+2
INACNM EQU ARG+4
LACNM EQU ARG+1A
CODE EQU ARG+34
ERROR EQU ARG+36
LENGTH EQU ARG+38
FLAG EQU ARG+30

REF ENT\$M,RET\$M
REF S\$BIDT
REF S\$NEW

L1 TEXT 'SYSBID'
DATA 2
DATA L2

```

DATA LI
DEF SYEBID
SYEBID 15 7 4
DATA 10094
MOVB @LENGTH+1(BOT),@INACNM(BOT)
MOVB @LENGTH+1(BOT),@LACNM(BOT)
MOVB @PFLUND+1(BOT),R1
SWPB R1
MOVB @TASKID+1(BOT),R1
LI R2,8 LOAD DISPLACMENT OF R4 FROM R0 INTO R2
LI R4,INACNM
LI R5,LACNM
LI R6,0
A R9,R2 ADD ADDRESS OF R0(BOTTOM OF STACK)TO R2
A R9,R4 ADD ADDRESS OF R0 TO RELATIVE ADDR OF R4
MOV R4,@WRTIN+6 MOVE ADDRESS OF INACNM TO WRTIN SVC BLOCK
A R9,R5 ADD ADDRESS OF R0 TO RELATIVE ADDR OF R5
MOV R5,@WRTL+6
MOVB @FLAG+1(BOT),R3 SET "FLAG" VALUE IN RIGHT BYTE OF R3
SWPB R3
MOVB @CODE+1(BOT),R3 SET "CODE" VALUE IN LEFT BYTE OF R3
BLWP @S$NEW
BLWP @S$BIDT
MOV @ERROR(BOT),R3 MOV ADDRESS OF ERROR INTO R3
MOV R0,*R3
XOP @OPEN,15
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCR,15
MOV R1,R0
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCR,15
MOV R2,R0
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCR,15
MOV R3,R0
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCR,15
MOV R4,R0
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCR,15
MOV R5,R0
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCR,15
MOV R6,R0
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCR,15
MOV R9,R0
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCR,15
MOV @TASKID(R9),R0
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCR,15
MOV @PFLUND(R9),R0

```

```

XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCL,15
MOV @CODE(RR),R0
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCL,15
MOV @ERROR(RR),R3      MOVE ADDRESS OF ERROR INTO R3
MOV *R3,R0             MOVE ERROR VALUE INTO R0
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCL,15
MOV @LENGTH(RR),R0
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCL,15
MOV @FLAG(RR),R0
XOP @BINHEX,15
XOP @WRTHEX,15
XOP @WRTCL,15

```

```

XOP @WRTIN,15
XOP @WRTCL,15
XOP @WRTL,15
XOP @WRTCL,15

```

```

L2      B   @RET#M
        END

```

#20

#0

AFB500.MCLEOD.XB.SHOW

AFB500.MCLEOD.XB.LIST

#0

#0

BATCH
.SHOW 4P5500.MOLEOD.3AC.P4DATA
EBATCH

*ECIPPO ** ECIPPO ** ECIPPO ** ECIPPO ** ECIPPO ** ECIPPO ** *

<0001> BATCH
LS (LIST SYNONYMS) ? NO
BATCH LISTING ACCESS NAME ** NULL **
BATCH INPUT ACCESS NAME ** NULL **
STATION ID ST03
USER ID TMM012
17:11:53 MONDAY, SEP 08, 1980.

<0002> .SHOW AFB500.MCLEOD.SRC.P4DATA

#20

#0

AFB500.MCLEOD.XB.SHOW

AFB500.MCLEOD.XB.LIST

#0

#0

<0003> EBATCH

CODE ** NULL **

TEXT ** NULL **

LS (LIST SYNONYMS) ? NO

Vita


Thomas M. McLeod was born on 30 June 1945 in Madison, Wisconsin. He graduated from high school in Oxon Hill, Maryland in 1963 and was employed by American Telegraph and Telephone until he entered the Navy in July, 1964. During three of four years of active duty in the Navy, he served as an Aviation Firecontrol Technician in the aircraft carrier, U.S.S. Forrestal (CVA-59). Upon release from active duty in June, 1968, he began 6 years in the active Naval Air Reserve and returned to AT&T. He was employed by the U.S. National Bureau of Standards from April, 1969 to January, 1973 as an electronics technician (GS-8) and by Fairchild Space and Electronics Company from January, 1973 to September, 1974 as an Associate Engineer. Upon Graduation from the University of Maryland in 1974, he was employed as an electronics engineer at Singer Simulation Inc. until entering the Air Force in 1975. He was commissioned from Officer Training School in May, 1975 and assigned to the Space and Missile Systems Organization, Los Angeles AFS, CA as a Communications Satellite Engineer. He entered the School of Engineering, Air Force Institute of Technology, in June, 1979.

Permanent address: 4518 S. Lois Ave.

Tampa, FL 33611

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GE/EE/80D-30	2. GOVT ACCESSION NO. AD A100793	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Design of a Resource-Sharing Network Link		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Thomas M. McLeod Captain USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Aeronautical Systems Division (ASD/ADDI) Wright-Patterson AFB, Ohio 45433		12. REPORT DATE December 1980
		13. NUMBER OF PAGES 119
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) 11 JUN 1981		
18. SUPPLEMENTARY NOTES Approved for public release; IAW AFR 190-17 Frederic C. Lynch, Major, USAF Director of Public Affairs <div style="text-align: right;">  FREDRIC C. LYNCH, Major, USAF Director of Public Affairs </div>		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Minicomputers Computer Networks Resource-Sharing Protocols Computer Interfaces <div style="text-align: right;"> Air Force Institute of Technology (AFIT) Wright-Patterson AFB, OH 45433 </div>		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A resource sharing (R-S) computer network link design is presented. The design complies with requirements for the Automated Management System (AMS) being developed by the Air Force's Aeronautical Systems Division (ASD). The design includes Texas Instruments Model 990 computers that currently exist in the AMS Network. R-S software was developed that binds existing modules into, effectively, one R-S process at each end of the link. The two R-S processes cooperate in		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(Continued from block 20)

accordance with R-S protocols to accomplish the specified R-S functions. Existing capabilities of the DX-10 operating system were used extensively. The processes communicate via telephone circuits using TI 3780 Emulator software and standard data communications hardware. The R-S functions implemented include file transfer, program file transfer, and remote program execution, as well as others.

A unique interface to the DX-10 operating system was designed to allow the R-S software to have adequate access to the operating system. The operating system batch command was emulated, allowing the R-S software to construct a "batch file" containing any operating system commands that would normally be entered from an interactive terminal.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)